

C3

[illegible][illegible]

Function Index Page 1

Fri Jan 04 16:35:25 2008 Function Index Page 2


```

83 RSSTable.c 1
84 RSSTable.c 13
85 RSSTable.c 18
86 RSSTable.c 25
87 RSSTable.c 38
88 RSSTable.c 40
89 RSSTable.c 42
90 RSSTable.c 44
91 RSSTable.c 46
92 RSSTable.c 48
93 RSSTable.c 50
94 RSSTable.c 52
95 RSSTable.c 54
96 RSSTable.c 56
97 RSSTable.c 58
98 RSSTable.c 60
99 RSSTable.c 62
100 RSSTable.c 64
101 RSSTable.c 66
102 RSSTable.c 68
103 RSSTable.c 70
104 RSSTable.c 72
105 RSSTable.c 74
106 RSSTable.c 76
107 RSSTable.c 78
108 RSSTable.c 80
109 RSSTable.c 82
110 RSSTable.c 84
111 RSSTable.c 86
112 RSSTable.c 88
113 RSSTable.c 90
114 RSSTable.c 92
115 RSSTable.c 94
116 RSSTable.c 96
117 RSSTable.c 98
118 RSSTable.c 100
119 RSSTable.c 102
120 RSSTable.c 104
121 RSSTable.c 106
122 RSSTable.c 108
123 RSSTable.c 110
124 RSSTable.c 112
125 RSSTable.c 114
126 RSSTable.c 116
127 RSSTable.c 118
128 RSSTable.c 120
129 RSSTable.c 122
130 RSSTable.c 124
131 RSSTable.c 126
132 RSSTable.c 128
133 RSSTable.c 130
134 RSSTable.c 132
135 RSSTable.c 134
136 RSSTable.c 136
137 RSSTable.c 138
138 RSSTable.c 140
139 RSSTable.c 142
140 RSSTable.c 144
141 RSSTable.c 146
142 RSSTable.c 148
143 RSSTable.c 150
144 RSSTable.c 152
145 RSSTable.c 154
146 RSSTable.c 156
147 RSSTable.c 158
148 RSSTable.c 160
149 RSSTable.c 162
150 RSSTable.c 164
151 RSSTable.c 166
152 RSSTable.c 168
153 RSSTable.c 170
154 RSSTable.c 172
155 RSSTable.c 174
156 RSSTable.c 176
157 RSSTable.c 178
158 RSSTable.c 180
159 RSSTable.c 182
160 RSSTable.c 184
161 RSSTable.c 186
162 RSSTable.c 188
163 RSSTable.c 190
164 RSSTable.c 192
165 RSSTable.c 194
166 RSSTable.c 196
167 RSSTable.c 198
168 RSSTable.c 200
169 RSSTable.c 202
170 RSSTable.c 204
171 RSSTable.c 206
172 RSSTable.c 208
173 RSSTable.c 210
174 RSSTable.c 212
175 RSSTable.c 214
176 RSSTable.c 216
177 RSSTable.c 218
178 RSSTable.c 220
179 RSSTable.c 222
180 RSSTable.c 224
181 RSSTable.c 226
182 RSSTable.c 228
183 RSSTable.c 230
184 RSSTable.c 232
185 RSSTable.c 234
186 RSSTable.c 236
187 RSSTable.c 238
188 RSSTable.c 240
189 RSSTable.c 242
190 RSSTable.c 244
191 RSSTable.c 246
192 RSSTable.c 248
193 RSSTable.c 250
194 RSSTable.c 252
195 RSSTable.c 254
196 RSSTable.c 256
197 RSSTable.c 258
198 RSSTable.c 260
199 RSSTable.c 262
200 RSSTable.c 264
201 RSSTable.c 266
202 RSSTable.c 268
203 RSSTable.c 270
204 RSSTable.c 272
205 RSSTable.c 274
206 RSSTable.c 276
207 RSSTable.c 278
208 RSSTable.c 280
209 RSSTable.c 282
210 RSSTable.c 284
211 RSSTable.c 286
212 RSSTable.c 288
213 RSSTable.c 290
214 RSSTable.c 292
215 RSSTable.c 294
216 RSSTable.c 296
217 RSSTable.c 298
218 RSSTable.c 300
219 RSSTable.c 302
220 RSSTable.c 304
221 RSSTable.c 306
222 RSSTable.c 308
223 RSSTable.c 310
224 RSSTable.c 312
225 RSSTable.c 314
226 RSSTable.c 316
227 RSSTable.c 318
228 RSSTable.c 320
229 RSSTable.c 322
230 RSSTable.c 324
231 RSSTable.c 326
232 RSSTable.c 328
233 RSSTable.c 330
234 RSSTable.c 332
235 RSSTable.c 334
236 RSSTable.c 336
237 RSSTable.c 338
238 RSSTable.c 340
239 RSSTable.c 342
240 RSSTable.c 344
241 RSSTable.c 346
242 RSSTable.c 348
243 RSSTable.c 350
244 RSSTable.c 352
245 RSSTable.c 354
246 RSSTable.c 356
247 RSSTable.c 358
248 RSSTable.c 360
249 RSSTable.c 362
250 RSSTable.c 364
251 RSSTable.c 366
252 RSSTable.c 368
253 RSSTable.c 370
254 RSSTable.c 372
255 RSSTable.c 374
256 RSSTable.c 376
257 RSSTable.c 378
258 RSSTable.c 380
259 RSSTable.c 382
260 RSSTable.c 384
261 RSSTable.c 386
262 RSSTable.c 388
263 RSSTable.c 390
264 RSSTable.c 392
265 RSSTable.c 394
266 RSSTable.c 396
267 RSSTable.c 398
268 RSSTable.c 400
269 RSSTable.c 402
270 RSSTable.c 404
271 RSSTable.c 406
272 RSSTable.c 408
273 RSSTable.c 410
274 RSSTable.c 412
275 RSSTable.c 414
276 RSSTable.c 416
277 RSSTable.c 418
278 RSSTable.c 420
279 RSSTable.c 422
280 RSSTable.c 424
281 RSSTable.c 426
282 RSSTable.c 428
283 RSSTable.c 430
284 RSSTable.c 432
285 RSSTable.c 434
286 RSSTable.c 436
287 RSSTable.c 438
288 RSSTable.c 440
289 RSSTable.c 442
290 RSSTable.c 444
291 RSSTable.c 446
292 RSSTable.c 448
293 RSSTable.c 450
294 RSSTable.c 452
295 RSSTable.c 454
296 RSSTable.c 456
297 RSSTable.c 458
298 RSSTable.c 460
299 RSSTable.c 462
300 RSSTable.c 464
301 RSSTable.c 466
302 RSSTable.c 468
303 RSSTable.c 470
304 RSSTable.c 472
305 RSSTable.c 474
306 RSSTable.c 476
307 RSSTable.c 478
308 RSSTable.c 480
309 RSSTable.c 482
310 RSSTable.c 484
311 RSSTable.c 486
312 RSSTable.c 488
313 RSSTable.c 490
314 RSSTable.c 492
315 RSSTable.c 494
316 RSSTable.c 496
317 RSSTable.c 498
318 RSSTable.c 500
319 RSSTable.c 502
320 RSSTable.c 504
321 RSSTable.c 506
322 RSSTable.c 508
323 RSSTable.c 510
324 RSSTable.c 512
325 RSSTable.c 514
326 RSSTable.c 516
327 RSSTable.c 518
328 RSSTable.c 520
329 RSSTable.c 522
330 RSSTable.c 524
331 RSSTable.c 526
332 RSSTable.c 528
333 RSSTable.c 530
334 RSSTable.c 532
335 RSSTable.c 534
336 RSSTable.c 536
337 RSSTable.c 538
338 RSSTable.c 540
339 RSSTable.c 542
340 RSSTable.c 544
341 RSSTable.c 546
342 RSSTable.c 548
343 RSSTable.c 550
344 RSSTable.c 552
345 RSSTable.c 554
346 RSSTable.c 556
347 RSSTable.c 558
348 RSSTable.c 560
349 RSSTable.c 562
350 RSSTable.c 564
351 RSSTable.c 566
352 RSSTable.c 568
353 RSSTable.c 570
354 RSSTable.c 572
355 RSSTable.c 574
356 RSSTable.c 576
357 RSSTable.c 578
358 RSSTable.c 580
359 RSSTable.c 582
360 RSSTable.c 584
361 RSSTable.c 586
362 RSSTable.c 588
363 RSSTable.c 590
364 RSSTable.c 592
365 RSSTable.c 594
366 RSSTable.c 596
367 RSSTable.c 598
368 RSSTable.c 600
369 RSSTable.c 602
370 RSSTable.c 604
371 RSSTable.c 606
372 RSSTable.c 608
373 RSSTable.c 610
374 RSSTable.c 612
375 RSSTable.c 614
376 RSSTable.c 616
377 RSSTable.c 618
378 RSSTable.c 620
379 RSSTable.c 622
380 RSSTable.c 624
381 RSSTable.c 626
382 RSSTable.c 628
383 RSSTable.c 630
384 RSSTable.c 632
385 RSSTable.c 634
386 RSSTable.c 636
387 RSSTable.c 638
388 RSSTable.c 640
389 RSSTable.c 642
390 RSSTable.c 644
391 RSSTable.c 646
392 RSSTable.c 648
393 RSSTable.c 650
394 RSSTable.c 652
395 RSSTable.c 654
396 RSSTable.c 656
397 RSSTable.c 658
398 RSSTable.c 660
399 RSSTable.c 662
400 RSSTable.c 664
401 RSSTable.c 666
402 RSSTable.c 668
403 RSSTable.c 670
404 RSSTable.c 672
405 RSSTable.c 674
406 RSSTable.c 676
407 RSSTable.c 678
408 RSSTable.c 680
409 RSSTable.c 682
410 RSSTable.c 684
411 RSSTable.c 686
412 RSSTable.c 688
413 RSSTable.c 690
414 RSSTable.c 692
415 RSSTable.c 694
416 RSSTable.c 696
417 RSSTable.c 698
418 RSSTable.c 700
419 RSSTable.c 702
420 RSSTable.c 704
421 RSSTable.c 706
422 RSSTable.c 708
423 RSSTable.c 710
424 RSSTable.c 712
425 RSSTable.c 714
426 RSSTable.c 716
427 RSSTable.c 718
428 RSSTable
```

```

RSTgetObj.c ..... 1
    RSTL_GetTopLevelObjects..... 3
RSLGetRoba.c ..... 13
    GetTLContents..... 18
        RSTL_GetResourceObjects 15
..//lib_rsrcore/RILFunc.c 25
    CMLC ..... 36
        RSTL_FullName..... 38
        FindBaseIndex..... 43
        GetClientTypeFromConfig..... 47
        RSTL_AddVolumeToRstList 48
        RSTL_FillRstObj..... 51
        RSTL_FreeOldIdList ..... 54
        RSTL_GetDirContents..... 29
        RSTL_RemoveVolumeFromList ..... 50
        RSTL_SetWorkItem..... 26
            Fill_Client_dirtop2 ..... 45
..//lib_rsrcore/RILData.c 53
    RSTL_MarkObj..... 80
        RSTL_UnmarkObj..... 69
        RSTL_UnmarkObj..... 75
        allowed_to_mark..... 66
        check_parent_perms ..... 68
        mark_tree..... 60
        mtc ..... 58
        umtx..... 59
            unmark_tree ..... 64
RSLSubmt.c ..... 81
    RSTL_Submit ..... 83
        RSTL_GetLogInfo..... 102
        RSTL_GetLogInfo..... 100
        fill_client_dirtop..... 91
        push_info_to_submitfile 95
        push_submt_file..... 93
        push_to_submitfile ..... 99
        send2bfd..... 101
        send2bfd..... 109
..//lib_rsrcore/SubmtFile.c 104
    CreateSubmtFile..... 110
    OpenSubmtFile..... 112
    Read ..... 121
    ReadBitFromFileForSubmtFile..... 122
    Write ..... 120
        WriteToFileInForsubmtfile..... 117

GetSEIOVolumeName ..... 139
GetSEIOKeyName ..... 146
GetSEIOKeyType ..... 160
GetSEBackupAdmin ..... 162
GetSEInstallationAdmin ..... 164
GetSEEffectivelID ..... 166
GetSEEffectiveUser Name ..... 168
GetSEAccessionPhase ..... 169
GetSEAccessionary ..... 169
GetSEPostPhase ..... 172
GetSEPrephase ..... 174
GetSESourceSystemAdmin ..... 163
GetSEUserID ..... 165
GetSEUserName ..... 167
GetSEVolumeNameNeeded ..... 170
GetSEWorkItemCount ..... 171
IsValidSubmtCID ..... 175
IsValidSubmtElement ..... 177
LookupSubmtObject ..... 142
NewSubmtElement ..... 144
NewSubmtFiles ..... 143
RemoveSubmtFiles ..... 140
SetSEBasics ..... 185
SetSEDestination ..... 186
SetSEDType ..... 187
SetSEBConnect ..... 189
SetSEBugReport ..... 193
SetSEDeviceName ..... 192
SetSESysTime ..... 192
SetSESummary ..... 190
SetSEVolume ..... 191
SetSOAdminID ..... 181
SetSOBasics ..... 176
SetSOPostPhase ..... 179
SetSOPrePhase ..... 180
SetSOFirePhase ..... 178
SetSOTotalSize ..... 182
SetSOTimes ..... 182
SetSOTimeID ..... 182
SetSOWMCheck ..... 177
LockSubmtDataEx ..... 137
UnlockSubmtDataEx ..... 139

..//lib_rsrcore/EIWRSubmtElemnt.c 221

```



```

2  /*.....
3  **
4  ** File Name:  RSigetLib.c
5  **
6  ** Copyright (c) 1998, 1999 by EMC Corporation.
7  **
8  ** Purpose:
9  **   This module contains the RSISL.GettopLevelObjects
10  **   Restore Service Library Function.
11  **   This function is provided to allow retrieval of the
12  **   top level objects which are restorable for the given client.
13  **
14  **
15  ** Compile-Time Options:
16  **   This section must list any compile time definitions
17  **   which will affect this header.
18  **
19  ****/

```

```

22  /* The following provides an RCS id in the binary that can be located
23  ** with the What(1) utility.  The intent is to keep this short.
24  */

```

```

26  #ifndef lint
27  static char RCS_id [] = "SRCFILES"
28  " $Revision$ "
29  " $Date$ " ;
30  #endif

```

```

33  /*
34  ** Feature test switches.
35  ** Standard defines required to turn on OS features go here.
36  *
37  * The following is required for code that uses POSIX APIs.
38  * Remove for non-POSIX, non-portable code.
39  */

```

```

41  #define _POSIX_SOURCE 1

```

```

44  /*
45  ** System headers.
46  */

```

```

49  /*
50  ** Epoch headers.
51  */
52  #include <ob/ob_port.h>
53  #include <ob/rb_log.h>

```

```

56  /*
57  ** Local headers
58  */
59  #include <RSIinterns.h>

```

```

62  /*
63  ** #defines, structures, typedefs local to this source file
64  */

```

```

66  typedef struct wi_info
67  {
68  1   struct wi_info *next;
69  2   char *wi_name;
70  3   char *wi_work;
71  4   char *wi_dir;
72  5   char wi_type;
73  6   } wi_info;

```

```

75  NEW_SINc_FILE();

```

```

77  /*
78  ** External declarations
79  */

```



```

206 2 while (NULL != top_wl_list)
207 3 {
208 4     if (NULL != top_wl_list->wl_name)
209 5     {
210 6         free(top_wl_list->wl_name);
211 7     }
212 8     if (NULL != top_wl_list->wl_work)
213 9     {
214 10         free(top_wl_list->wl_work);
215 11     }
216 12     if (NULL != top_wl_list->wl_bic)
217 13     {
218 14         free(top_wl_list->wl_bic);
219 15     }
220 16     tmp_list = top_wl_list->next;
221 17     free(top_wl_list);
222 18     top_wl_list = tmp_list;
223 19 }
224 20
225 21 /* Free unused top level objects from plugins */
226 22 if (pluginlist)
227 23 {
228 24     RSTSL_FreeRestorableObjectList(pluginlist);
229 25     pluginlist = NULL;
230 26 }
231 27
232 28 /* Scan all the workitems in all the workgroups
233 29 * We are going to get all of the items that meet the
234 30 criteria
235 31 * and keep them in static memory. Further calls will feed off
236 32 * of this list.
237 33 */
238 34 for (wgp = tcp->rc.config->pgrouplist;
239 35      wgp != NULL;
240 36      wgp = wgp->next)
241 37 {
242 38     RBC_MONKITEM *wip;
243 39     for (wip = wgp->wpllist; wip != NULL; wip = wip->next)
244 40     {
245 41         /* host must match and wctype must not be a plug-in's
246 42          *
247 43          * if ( 0 == strcmp(wip->sysname, sourcehost))
248 44          * && ((exechcode == RAW_NETWORK)
249 45             || NULL == tcp->rc.num.plugin.wl_types
250 46             || tcp->rc.plugin.wl_types == wip->wl_type)
251 47             tcp->rc.num.plugin.wl_types)
252 48         {
253 49             n_node = linked_list_new(
254 50                 generic_list_cpy, *);
255 51             if (n_node == NULL)
256 52                 break;
257 53             result = EP_RB_RECOVER_NOMEM;
258 54             n_node->wl_name = eal_strdup(wip->name);
259 55             if (NULL == n_node->wl_name)
260 56                 break;
261 57             result = EP_RB_RECOVER_NOMEM;
262 58             break;
263 59 }
264 60 }

```

```

265 61 /* For striped workitems, the work item list can be
266 62 * NULL, for the stripes > 1 of workitems without a
267 63 * partitionspec.
268 64 */
269 65 if (
270 66     NULL != top_wl_list) /* else wl_work already NULL */
271 67 {
272 68     n_node->wl_work = eal_strdup(wip->list);
273 69     if (NULL == n_node->wl_work)
274 70     {
275 71         result = EP_RB_RECOVER_NOMEM;
276 72         break;
277 73     }
278 74 }
279 75
280 76 if (NULL != wip->backup_init)
281 77 {
282 78     n_node->wl_bic = eal_strdup(wip->backup_init);
283 79     if (NULL == n_node->wl_bic)
284 80     {
285 81         result = EP_RB_RECOVER_NOMEM;
286 82         break;
287 83     }
288 84 }
289 85 n_node->wl_type = wip->wl_type;
290 86
291 87 } /* end inner for scanning workitems within a workgroup */
292 88 } /* end inner for scanning workitems within a workgroup */
293 89 result = E_SUCCESS /* malloc failure, break out */
294 90 break;
295 91 } /* end outer for scanning workgroups */
296 92
297 93 /* Save the pointer to the head of the list for freeing later.
298 94 */
299 95 top_wl_list = wl_list;
300 96
301 97 if (result != E_SUCCESS) {
302 98     if (result == EP_RB_RECOVER_NOMEM) {
303 99         rec_apl_log_cm(SUB_CSN_NOMEM, NULL);
304 100     }
305 101     /* free wl_list next time in */
306 102     return result;
307 103 }
308 104
309 105 /* Save the pointer to the head of the list for freeing later.
310 106 */
311 107 top_wl_list = wl_list;
312 108
313 109 if (result != E_SUCCESS) {
314 110     if (result == EP_RB_RECOVER_NOMEM) {
315 111         rec_apl_log_cm(SUB_CSN_NOMEM, NULL);
316 112     }
317 113     /* free wl_list next time in */
318 114     return result;
319 115 }
320 116
321 117 if (exechcode != RAW_NETWORK)
322 118     /* save appdata pointer in case a cto is selected already */
323 119     saved_appdata = tcp->appdata;
324 120
325 121 /* get top level object list(s) from plugin(s) */
326 122 for (index = 1, pldict = tcp->pldict;
327 123      index++, pldict = pldict->next)
328 124 }

```



```

318 4         {
319 4             cliOpt = NULL;
320 4             tmp_count = 0;
321 4             tcp->appdata = pIpPr->appdata;
322 4             result = pIpPr->pfFuncArray[pFuncIndex&0x7f]
323 4         }
324 4         if ( result != E_SUCCESS,
325 4             || (NULL == cliOpt && tmp_count > 0)
326 4             || (NULL == cliOpt && tmp_count == 0) )
327 4         {
328 4             /* error or inconsistent results */
329 4             result =
330 4                 "The internal error";
331 4             *pPlugin = &g;
332 4             error in GetTopLevelObjects call",
333 4             (struct pluginInData *){
334 4                 pIpPr->appdata->name
335 4             };
336 4             result = E_SUCCESS; /* not fatal */
337 4             continue;
338 4         }
339 4         if (NULL == cliOpt) /* none from this plug-in */
340 4             continue;
341 4         if (NULL == pluginInList)
342 4             pluginInList = cliOpt;
343 4         else
344 4             cliOpt->next = cliOpt;
345 4         for ( ; cliOpt && tmp_count;
346 4             cliOpt->next = cliOpt, tmp_count--, pI_count++)
347 4         {
348 4             cliOpt->cli->root.backupApp = index;
349 4             cliOpt = cliOpt;
350 4             cliOpt = cliOpt->next;
351 4         }
352 4         if (
353 4             tmp_count || cliOpt) /* inconsistent output */
354 4             cliOpt->internal_error(
355 4                 "plugin %s:
356 4                 bad output from GetTopLevelObjects call",
357 4                 (struct pluginInData *){
358 4                     pIpPr->appdata->name
359 4                 }
360 4             );
361 4         tcp->appdata = saved_appdata; /* restore saved appdata ptr */
362 4         else
363 4         {
364 4             pluginInList = NULL;
365 4             pI_count = 0;
366 4         }
367 4         else if ((valid_cookie != *cookie) || (DONE_COOKIE == *cookie))
368 4         {
369 4             return(EP_RB_RECOVER_AND_COOKIE);
370 4         }
371 4     }
372 4 }
373 4
374 4
375 4
376 4
377 4
378 4
379 4
380 4
381 4
382 4
383 4

```

```

385 1         /*
386 1         * Logic drops through to here regardless of the passed in cookie
387 1         * being the INIT state or a valid key in a subsequent call.
388 1         */
389 1         /*
390 1         * Return a linked list of the number of "restorable objects"
391 1         * requested.
392 1         * Stop if we reach the end of the list.
393 1         */
394 1         /* set linked list and current entry pointers to NULL at start */
395 1         *topLevelObjs = cliOpt = NULL;
396 1         index = 0;
397 1         while ( (total_count <= 0, count + pI_count)
398 1             || (cliOpt == NULL)
399 1             && (wI_list != NULL) || (pPluginList != NULL) )
400 1         {
401 1             if (wI_list != NULL)
402 1             {
403 1                 cliOpt = linked_list_new( {
404 1                     generic_list_by ** topLevelObjs,
405 1                     sizeof( struct RSTRPC_tlo_list ),
406 1                     struct RSTRPC_tlo_list );
407 1                 if ( (NULL == cliOpt)
408 1                     || (cliOpt->tlo = calloc(1, sizeof(
409 1                         struct RSTRPC_top_level_obj) )
410 1                         == NULL) )
411 1                 {
412 1                     /* do we return partial list, or free list & return none ? */
413 1                     /* need a local function to free list of restorable on error */
414 1                     result = EP_RB_RECOVER_NOMEM;
415 1                     break;
416 1                 }
417 1                 cliOpt->tlo->root.objLevel = RSTRPC_tlo_type;
418 1                 cliOpt->tlo->root.objName = &sl_strdup(wI_list->wI_name);
419 1                 if ( (NULL == cliOpt->tlo->root.objName)
420 1                     || (NULL == cliOpt->tlo->root.objName) )
421 1                 {
422 1                     result = EP_RB_RECOVER_NOMEM;
423 1                     break;
424 1                 }
425 1                 cliOpt->tlo->root.backupApp = 0; /* value for network objects */
426 1                 if (
427 1                     NULL != wI_list->wI_work) /* may now be null for oldb kicker */
428 1                 {
429 1                     cliOpt->tlo->fileSpec = &sl_strdup(wI_list->wI_work);
430 1                     if ( (NULL == cliOpt->tlo->fileSpec)
431 1                         || (NULL == cliOpt->tlo->fileSpec) )
432 1                     {
433 1                         result = EP_RB_RECOVER_NOMEM;
434 1                         break;
435 1                     }
436 1                 }
437 1                 if ( (NULL != wI_list->wI_bic)
438 1                     || (cliOpt->tlo->wIbic = &sl_strdup(wI_list->wI_bic);
439 1                         if ( (NULL == cliOpt->tlo->wIbic)
440 1                             || (NULL == cliOpt->tlo->wIbic) )
441 1                         {
442 1                             result = EP_RB_RECOVER_NOMEM;
443 1                             break;
444 1                         }
445 1                     }
446 1                 }
447 1             }
448 1         }
449 1     }
450 1 }
451 1
452 1
453 1
454 1
455 1
456 1
457 1
458 1
459 1
460 1
461 1

```

```

444 3      )
445 3      }
446 3      clofter->plio->w1_type = w1_list->w1_type;
447 3      }
448 3      w1_list = w1_list->next;
449 3      }
450 3      else /* get c1o from plugin list: */
451 3      {
452 3          if (NULL == *toplevobj)
453 3          {
454 3              /* set start of list ptr to plugin c1o's */
455 3              *toplevobj = pluginlist;
456 3          }
457 3          else /* link prev entry to this one in plugin list */
458 3          {
459 3              clofter->next = pluginlist;
460 3              clofter->plio->next = pluginlist;
461 3              pluginlist = pluginlist->next;
462 3              clofter->next = NULL;
463 3          }
464 3      }
465 3      /* set hostname in all top level objects */
466 3      clofter->plio->hostname = eal_strdup( sourcehost );
467 3      if (NULL == clofter->hostname)
468 3      {
469 3          result = RP_RB_RECOVER_NONMEM;
470 3          break;
471 3      }
472 3      ++count;
473 3      }
474 3      }
475 3      }
476 3      /* catch all malloc failures here: free list, return failure /
477 3      nothing) */
478 3      if (result != R_SUCCESS) {
479 3          rec_api_log_cmm(SUB_CSM_NONMEM, NULL);
480 3          index = 0;
481 3      }
482 3      /* to indicate none returned
483 3      valid cookie = INTF_COOKIE;
484 3      total_count = w1_count + pl_count /* to cause free of w1 lists */
485 3      /* free linked list output */
486 3      if (*toplevobj != NULL)
487 3      {
488 3          RSTSL_FreeRestorableObjList( *toplevobj );
489 3          *toplevobj = NULL;
490 3      }
491 3      *numberEntities = index;
492 3      /*
493 3      * set the cookie appropriately.
494 3      * Also store it in the static 'valid cookie'
495 3      */
496 3      if (w1_count + pl_count >= total_count) /* any more TLOs for
497 3      later? */
498 3      {
499 3          /* yes: */
500 3          if (
501 3              w1_count >= total_count) { /* any more netwk TLOs? */
502 3              *cookie = (long)w1_list; /* yes, still have netwk WTS */
503 3          }
504 3      }

```

```

505 3      else {
506 3          *cookie = (long)pluginlist;
507 3          /* no, rest are plugin TLOs */
508 3      }
509 3      valid cookie = *cookie;
510 3      }
511 3      else
512 3      {
513 3          /*
514 3          * Set the cookie to DONE state and the valid cookie to match
515 3          it.
516 3          */
517 3          if (result == R_SUCCESS)
518 3          {
519 3              /* dont set cookie on errors */
520 3              *cookie = DONE_COOKIE;
521 3              valid_cookie = DONE_COOKIE;
522 3          }
523 3          /*
524 3          * Free up memory. We are done and don't need it anymore.
525 3          * Start from the top of the list.
526 3          */
527 3          while (NULL != top_w1_list)
528 3          {
529 3              if (NULL != top_w1_list->w1_name)
530 3              {
531 3                  free(top_w1_list->w1_name);
532 3              }
533 3              if (NULL != top_w1_list->w1_work)
534 3              {
535 3                  free(top_w1_list->w1_work);
536 3              }
537 3              if (NULL != top_w1_list->w1_bic)
538 3              {
539 3                  free(top_w1_list->w1_bic);
540 3              }
541 3              top_w1_list = top_w1_list->next;
542 3          }
543 3          free(top_w1_list);
544 3          top_w1_list = tmp_list;
545 3          if (pluginlist)
546 3          {
547 3              RSTSL_FreeRestorableObjList( pluginlist );
548 3              pluginlist = NULL;
549 3          }
550 3          return( result );
551 3          /* end of RSTSL_GetTopLevelObjects() */

```



```

2  /*****
3  **
4  ** File Name:  RSIGetrcbs.c
5  **
6  ** Copyright (c) 1998, 1999 by EMC Corporation.
7  **
8  **
9  **
10 **
11 **
12 **
13 ** Table of Contents:
14 ** -----
15 **
16 **
17 **
18 **
19 ** Internal Functions:
20 **
21 **
22 **
23 **
24 **
25 **
26 **
27 **
28 **
29 ** The following provides an RCS id in the binary that can be located
30 ** with the what() utility.  The intent is to keep this short.
31 **
32 **
33 #ifndef lint
34 static char RCS_id [] = "RSCSfile$ "
35 " $Revision$ "
36 " $Date$ " ;
37 #endif
38
39
40
41
42
43
44
45
46
47
48 #define _POSIX_SOURCE 1
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92

```

```

64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92

```

```

/* Local headers
*/
#include <RSIntems.h>
#include <RSStartup.h>

/* #defines, structures, typedefs local to this source file
*/

/* External declarations
*/

NEW_SRC_FILE();

/* Local function prototypes
*/

static eserrno_t GetTIOContents(
    struct RSRRPC_top_level_obj *tlobj_ptr,
    struct RSRRPC_uio_list **obj_list_ptr,
    const long maxentries,
    struct RSRRPC_uio_list **obj_list_ptr,
    long *centricities,
    long *cookies);

```

Page 15 of 248	RSSTIL_GetRestorableObjects	Fri Jan 04 16:35:25 2008
94	/*	151 2
95	RSSTIL_GetRestorableObjects:	152 2
96	*	153 3
97	* This function is provided to allow retrieval of the	154 3
98	* child objects which are restorable given the parent object. The	155 2
99	* caller specifies the parent object and whether or not	156 1
100	* to include bad files.	157 2
101	*	158 2
102	* Performs the 'get' on the asynchronous thread of the Restores	159 2
103	* Engine.	160 2
104	*	161 2
105	* Parameters:	162 1
106	* parentPtr I) - the parent object; pointer to either a top level	163 1
107	* object or container object	164 1
108	* I) - specifies whether parentObject points to top level or	165 1
109	* objects	166 1
110	* O) - a pre-allocated pointer to return the list of	167 1
111	* objects	168 1
112	* cookie I/O) - a place holder for the list position	169 1
113	* maxEntities I) - the maximum number of objects to return	170 2
114	* entities O) - the real number of objects returned in the list	171 1
115	* allowBad I) - flag whether or not to include bad files	172 1
116	*	173 1
117	*****	174 1
118	externally RSSTIL_GetRestorableObjects)	175 1
119	*****	176 1
120	restorableObjectPtr parentPtr,	177 1
121	enum RSSTRC_ObjectLevel objectLevel,	178 1
122	struct RSSTRC_ufo_list *objects,	179 1
123	long cookie,	180 2
124	const long maxEntities,	181 2
125	long entities,	182 1
126	const boolean_t allowBad }	183 1
127	{	184 1
128	extern_t result;	185 1
129	struct RSSTRC_ufo_list *ufoList;	186 1
130	struct RSSTRC_top_level_obj_t *parentPtr;	187 1
131	struct RSSTRC_user_restorable_object *ufoPtr	188 1
132	= (struct RSSTRC_user_restorable_object *)parentPtr;	189 1
133	/*	190 1
134	* validate the input arguments.	191 2
135	/*	192 1
136	if ((NULL == parentPtr) (NULL == objects) (193 1
137	return(EP_RB_RECOVER_BAD_ARGS);	194 1
138	NULL == entities))	195 1
139	{	196 1
140	return(EP_RB_RECOVER_BAD_ARGS);	197 1
141	if (objectLevel == RSSTRC_tlo_type) {	198 1
142	{	199 1
143	if (tloPtr->root_objName == NULL)	200 1
144	{	201 1
145	return(EP_RB_RECOVER_INVALID_OBJNAME);	202 1
146	}	203 1
147	return(EP_RB_RECOVER_INVALID_OBJNAME);	204 1
148	}	205 1
149	else if (objectLevel == RSSTRC_container_type) {	206 1
150	if (ufoPtr->root_objName == NULL)	207 1
151	{	208 2
152	return(EP_RB_RECOVER_INVALID_OBJNAME);	209 2
153	}	210 2
154	return(EP_RB_RECOVER_INVALID_OBJNAME);	211 2
155	RSU_getobjs.c3	212 2
156	Fri Jan 04 16:35:25 2008	213 2

Page 16 of 248	RSSTIL_GetRestorableObjects	Fri Jan 04 16:35:25 2008
151	2	151 2
152	2	152 2
153	3	153 3
154	3	154 3
155	2	155 2
156	1	156 1
157	2	157 2
158	2	158 2
159	2	159 2
160	2	160 2
161	2	161 2
162	1	162 1
163	1	163 1
164	1	164 1
165	1	165 1
166	1	166 1
167	1	167 1
168	1	168 1
169	1	169 1
170	2	170 2
171	1	171 1
172	1	172 1
173	1	173 1
174	1	174 1
175	1	175 1
176	1	176 1
177	1	177 1
178	1	178 1
179	1	179 1
180	2	180 2
181	2	181 2
182	1	182 1
183	1	183 1
184	1	184 1
185	1	185 1
186	1	186 1
187	1	187 1
188	1	188 1
189	1	189 1
190	1	190 1
191	2	191 2
192	1	192 1
193	1	193 1
194	1	194 1
195	1	195 1
196	1	196 1
197	1	197 1
198	1	198 1
199	1	199 1
200	1	200 1
201	1	201 1
202	1	202 1
203	1	203 1
204	1	204 1
205	1	205 1
206	1	206 1
207	1	207 1
208	2	208 2
209	2	209 2
210	2	210 2
211	2	211 2
212	2	212 2
213	2	213 2
214	2	214 2
215	2	215 2
216	2	216 2
217	2	217 2
218	2	218 2
219	2	219 2
220	2	220 2
221	2	221 2
222	2	222 2
223	2	223 2
224	2	224 2
225	2	225 2
226	2	226 2
227	2	227 2
228	2	228 2
229	2	229 2
230	2	230 2
231	2	231 2
232	2	232 2
233	2	233 2
234	2	234 2
235	2	235 2
236	2	236 2
237	2	237 2
238	2	238 2
239	2	239 2
240	2	240 2
241	2	241 2
242	2	242 2
243	2	243 2
244	2	244 2
245	2	245 2
246	2	246 2
247	2	247 2
248	2	248 2
249	2	249 2
250	2	250 2
251	2	251 2
252	2	252 2
253	2	253 2
254	2	254 2
255	2	255 2
256	2	256 2
257	2	257 2
258	2	258 2
259	2	259 2
260	2	260 2
261	2	261 2
262	2	262 2
263	2	263 2
264	2	264 2
265	2	265 2
266	2	266 2
267	2	267 2
268	2	268 2
269	2	269 2
270	2	270 2
271	2	271 2
272	2	272 2
273	2	273 2
274	2	274 2
275	2	275 2
276	2	276 2
277	2	277 2
278	2	278 2
279	2	279 2
280	2	280 2
281	2	281 2
282	2	282 2
283	2	283 2
284	2	284 2
285	2	285 2
286	2	286 2
287	2	287 2
288	2	288 2
289	2	289 2
290	2	290 2
291	2	291 2
292	2	292 2
293	2	293 2
294	2	294 2
295	2	295 2
296	2	296 2
297	2	297 2
298	2	298 2
299	2	299 2
300	2	300 2
301	2	301 2
302	2	302 2
303	2	303 2
304	2	304 2
305	2	305 2
306	2	306 2
307	2	307 2
308	2	308 2
309	2	309 2
310	2	310 2
311	2	311 2
312	2	312 2
313	2	313 2
314	2	314 2
315	2	315 2
316	2	316 2
317	2	317 2
318	2	318 2
319	2	319 2
320	2	320 2
321	2	321 2
322	2	322 2
323	2	323 2
324	2	324 2
325	2	325 2
326	2	326 2
327	2	327 2
328	2	328 2
329	2	329 2
330	2	330 2
331	2	331 2
332	2	332 2
333	2	333 2
334	2	334 2
335	2	335 2
336	2	336 2
337	2	337 2
338	2	338 2
339	2	339 2
340	2	340 2
341	2	341 2
342	2	342 2
343	2	343 2
344	2	344 2
345	2	345 2
346	2	346 2
347	2	347 2
348	2	348 2
349	2	349 2
350	2	350 2
351	2	351 2
352	2	352 2
353	2	353 2
354	2	354 2
355	2	355 2
356	2	356 2
357	2	357 2
358	2	358 2
359	2	359 2
360	2	360 2
361	2	361 2
362	2	362 2
363	2	363 2
364	2	364 2
365	2	365 2
366	2	366 2
367	2	367 2
368	2	368 2
369	2	369 2
370	2	370 2
371	2	371 2
372	2	372 2
373	2	373 2
374	2	374 2
375	2	375 2
376	2	376 2
377	2	377 2
378	2	378 2
379	2	379 2
380	2	380 2
381	2	381 2
382	2	382 2
383	2	383 2
384	2	384 2
385	2	385 2
386	2	386 2
387	2	387 2
388	2	388 2
389	2	389 2
390	2	390 2
391	2	391 2
392	2	392 2
393	2	393 2
394	2	394 2
395	2	395 2
396	2	396 2
397	2	397 2
398	2	398 2
399	2	399 2
400	2	400 2
401	2	401 2
402	2	402 2
403	2	403 2
404	2	404 2
405	2	405 2
406	2	406 2
407	2	407 2
408	2	408 2
409	2	409 2
410	2	410 2
411	2	411 2
412	2	412 2
413	2	413 2
414	2	414 2
415	2	415 2
416	2	416 2
417	2	417 2
418	2	418 2
419	2	419 2
420	2	420 2
421	2	421 2
422	2	422 2
423	2	423 2
424	2	424 2
425	2	425 2
426	2	426 2
427	2	427 2
428	2	428 2
429	2	429 2
430	2	430 2
431	2	431 2
432	2	432 2
433	2	433 2
434	2	434 2
435	2	435 2
436	2	436 2
437	2	437 2
438	2	438 2
439	2	439 2
440	2	440 2
441	2	441 2
442	2	442 2
443	2	443 2
444	2	444 2
445	2	445 2
446	2	446 2
447	2	447 2
448	2	448 2
449	2	449 2
450	2	450 2
451	2	451 2
452	2	452 2
453	2	453 2
454	2	454 2
455	2	455 2
456	2	456 2
457	2	457 2
458	2	458 2
459	2	459 2
460	2	460 2
461	2	461 2
462	2	462 2
463	2	463 2
464	2	464 2
465	2	465 2
466	2	466 2
467	2	467 2
468	2	468 2
469	2	469 2
470	2	470 2
471	2	471 2
472	2	472 2
473	2	473 2
474	2	474 2
475	2	475 2
476	2	476 2
477	2	477 2
478	2	478 2
479	2	479 2
480	2	480 2
481	2	481 2
482	2	482 2
483	2	483

```

212 2
213 2
214 2
215 1
216 1
217 2
218 2
219 2
220 2
221 2
222 2
223 3
224 3
225 2
227 2
228 3
229 3
230 3
231 3
232 3
233 2
234 3
235 3
236 3
237 2
238 2
239 3
240 3
241 3
242 3
243 3
244 3
245 3
246 3
247 3
248 2
249 1
250 1
251 1
252 1
253 2
254 2
255 2
256 1
258 1
260 1
}
/* we already know it's a container */
/* Top level object must have been established before this func
* is called to list dir contents.
if (tcp->rc_top_level_object_name == NULL)
return EP_RR_RECOVER_INVALID;
if (NULL != tcp->currentDirptr)
result =
tcp->currentDirptr->funcArray[IPFuncIndexGetNilO]
(tcp,
uofptr,
RSTRNC_container_type,
objects,
maxEntries,
cEntries,
allowBuf );
else
{
result = RSTSL_GetDirContents( tcp,
tcp->root_objName,
allowBuf,
objects,
cEntries,
cookie );
}
}
/* if successful, mark all objects with proper backup app: */
if (E_SUCCESS == result)
{
for ( uoflist = *objects; uoflist != uoflist->next )
uoflist->uro->root.backupapp = tcp->rc_backup_app;
}
return result;
/* RSTSL_GetRestoreObjects */

```

```

263 1
264 1
265 1
266 1
267 1
268 1
269 1
270 1
271 1
272 1
273 1
274 1
275 1
276 1
277 1
278 1
279 1
280 1
281 1
282 1
283 1
284 1
286 1
287 1
288 1
289 1
290 1
291 1
292 1
293 1
294 1
295 1
296 1
297 1
298 1
299 1
300 1
301 1
302 1
303 1
304 1
305 1
306 1
307 1
308 1
309 1
310 1
311 1
312 1
313 1
314 1
315 1
316 1
317 1
318 1
}
/*
* The environment needs to be reset under the following
* circumstances:
* - it's the first time ever that a top level object is selected;
* - the caller wants to switch to a top level object different
* from
* what the env is set up for currently;
*
* The following criteria are used to determine if the caller
* wants to switch to a different top level object:
* - the object name specified in the restoreobjobject is
* different from what's kept in the restoreobj;
* - the object name specified in the restoreobjobject is
* different from what's kept in the restore_context;
* - the object name in the restoreobjobject is the same
* as what's in the restore_context, but the template or
* the trailset used is different;
* or the client host is different;
*/
static errno_t
GetTLOContents( struct RSTRNC_top_level_obj *tloptr,
const long maxEntries,
struct RSTRNC_uro_list **objlistptr,
long *cEntries,
long *cookie)
{
char *tloNameP = tloptr->root_objName;
char *tloNameP2 = tloptr->templateName;
char *tloNameP3 = tloptr->uroName;
errno_t rc;
boolean_t lvrset = FALSE;
int index;
/*
* The environment needs to be reset under the following
* circumstances:
* - it's the first time ever that a top level object is selected;
* - the caller wants to switch to a top level object different
* from
* what the env is set up for currently;
*
* The following criteria are used to determine if the caller
* wants to switch to a different top level object:
* - the object name specified in the restoreobjobject is
* different from what's kept in the restoreobj;
* - the object name specified in the restoreobjobject is
* different from what's kept in the restore_context;
* - the object name in the restoreobjobject is the same
* as what's in the restore_context, but the template or
* the trailset used is different;
* or the client host is different;
*/

```

```

320 1         if ( 0 != strcmp(
321 1             | | tcp->rc_hostname, tcp->rc_source_client_hostname ) )
322 2             {
323 2                 tcp->rc_backup_app != tloft->root_backupApp) )
324 3             {
325 3                 reset = TRUE; /* switch to a different application! */
326 4             }
327 5             else if ( (tcp->rc_top_level_object_name == NULL)
328 6             | | ( 0 != strcmp(tcp->rc_top_level_object_name, tloftNamep) ) )
329 7             {
330 8                 reset = TRUE; /* switch to a different object */
331 9             }
332 10            else
333 11            {
334 12                /*
335 13                 * rc_top_level_object_name is already set and it's the same
336 14                 * as the input top_level_obj_name, we need to further check
337 15                 * the template. If the template is the same, we must
338 16                 * then check the trailset usage. If any of these is
339 17                 * different between what's in the input restoreObj object
340 18                 * and what's in the restoreContext, we need to reset
341 19                 * the environment.
342 20                 */
343 21            }
344 22            if (tcp->rc_template != default)
345 23            {
346 24                rcTemplateName = NULL;
347 25            }
348 26            rcTemplateName = tcp->rc_template_name;
349 27            if ( NULL != tNamep )
350 28            {
351 29                if ( NULL == rcTemplateName )
352 30                {
353 31                    reset = TRUE;
354 32                }
355 33                else if ( strcmp(tNamep, rcTemplateName) /* diff templ */ )
356 34                {
357 35                    reset = TRUE;
358 36                }
359 37                else if ( tcp->rc_saveSetThread != tloft->setThread )
360 38                {
361 39                    reset = TRUE;
362 40                }
363 41            }
364 42            else if ( NULL != rcTemplateName )
365 43            {
366 44                reset = TRUE;
367 45            }
368 46            if (reset)
369 47            {
370 48                if ( !cookie != INTF_COOKIE )
371 49                {
372 50                    return(EP_RB_RECOVER_INVNVAL);
373 51                }
374 52            }
375 53            /* If last cli was other than a network backup object,
376 54             * let app clean up: */
377 55            if (tcp->rc_top_level_object_name != NULL)
378 56            {
379 57                if ( NULL != tcp->currentPiptr )
380 58                {
381 59                    rc =
382 60                    tcp->currentPiptr->pfFuncArray[PFFuncIndexClearRC]( tcp );
383 61                }
384 62            }
385 63            RSLgprintf( 7

```

```

386 4         if (rc != E_SUCCESS)
387 5             {
388 6                 the_internal_error( rc,
389 7                     "plugin: bg error in
390 8                     clearRestoreContext call",
391 9                     ( struct pluginData *) {
392 10                         tcp->currentPiptr->idData }->name );
393 11             }
394 12             tcp->currentPiptr->appData = tcp->appData;
395 13             /* save its data */
396 14             }
397 15             else
398 16             {
399 17                 /* for network backups, just clear the mark context: */
400 18                 resetMasks( tcp );
401 19             }
402 20             free( tcp->rc_top_level_object_name );
403 21             tcp->rc_top_level_object_name = NULL;
404 22             /*
405 23              * If we're switching to a new source host, we must free
406 24              * the space used for the previous source host name string
407 25              * before
408 26              * resetting it to the new.
409 27              */
410 28             if ( 0 != strcmp(
411 29                 | | tloft->hostname, tcp->rc_source_client_hostname ) )
412 30             {
413 31                 if ( NULL != tcp->rc_source_client_hostname )
414 32                 {
415 33                     free( tcp->rc_source_client_hostname );
416 34                 }
417 35                 if ( NULL == tcp->rc_source_client_hostname )
418 36                 {
419 37                     rc = api_log_get( SGN_CSM_NOWM, NULL );
420 38                     return( EP_RB_RECOVER_NOWM );
421 39                 }
422 40             }
423 41             if ( 0 == (tcp->rc_backup_app = tloft->root_backupApp) )
424 42             {
425 43                 /* new app is network: */
426 44                 tcp->appData = NULL;
427 45                 tcp->currentPiptr = NULL;
428 46                 else /* find app in plugin list,
429 47                  * set its appData and pluginData ptrs */
430 48                 {
431 49                     for ( index=1, tcp->currentPiptr = tcp->pluginList;
432 50                         index < tcp->rc_backup_app &&
433 51                         tcp->currentPiptr->next;
434 52                         index++ )
435 53                     {
436 54                         tcp->currentPiptr = tcp->currentPiptr->next;
437 55                         if ( index != tcp->rc_backup_app )
438 56                             /* got null ptr too early */
439 57                             the_internal_error( EP_RB_RECOVER_PLUGIN,
440 58                                 "plugin in list corrupted" );
441 59                         return( EP_RB_RECOVER_PLUGIN );
442 60                     }
443 61                 }
444 62             }
445 63             RSLgprintf( 8

```

File Jan 04 16:35:25 2008	GetTLContents	Page 21 of 248	File Jan 04 16:35:25 2008	GetTLContents	Page 22 of 248
<pre> 440 1 } 441 2 rcp->appdata = rcp->currentPiptr->appdata; 442 3 443 4 /* 444 5 * If we're switching to a different tlo, we must: 445 6 * unlock any previous work items and free the workitem name 446 7 */ 447 8 if (rcp->rc_top_level_object_name != NULL) 448 9 { 449 10 if (rcp->rc_have_wlock) 450 11 { 451 12 eh.unlock_object(452 13 eh.object(WORKITEM, rcp->rc_workitem_name, 453 14 eh.unlock_privacy), 454 15 rcp->rc_have_wlock = 0; 455 16 free(rcp->rc_workitem_name); 456 17 rcp->rc_workitem_name = NULL; 457 18 } 458 19 } 459 20 if (rcp->rc_template_name != NULL) 460 21 { 461 22 free(rcp->rc_template_name); 462 23 } 463 24 if (thname != NULL) 464 25 { 465 26 rcp->rc_template_name = eal_strdup(thname); 466 27 rcp->rc_template_defaulted = FALSE; 467 28 if (NULL != rcp->rc_template_name) 468 29 { 469 30 reg_api_log_cm(SUB_CSN_NOWEM, NULL); 470 31 return(EP_RB_RECOVER_NOWEM); 471 32 } 472 33 } 473 34 else 474 35 { 475 36 rcp->rc_template_name = NULL; 476 37 rcp->rc_template_defaulted = TRUE; 477 38 } 478 39 rcp->rc_saveset_thread = tloptr->asethread; 479 40 480 41 /* Do this last, 481 42 to make sure all startup's succeed: else leave 482 43 rcp->rc_top_level_object_name NULL */ 483 44 rcp->rc_top_level_object_name = eal_strdup(thname); 484 45 if (NULL == rcp->rc_top_level_object_name) 485 46 { 486 47 reg_api_log_cm(SUB_CSN_NOWEM, NULL); 487 48 return(EP_RB_RECOVER_NOWEM); 488 49 } 489 50 } 490 51 491 52 /* 492 53 * Check if restore is authorized to the specified client 493 54 * and by the specified user. 494 55 */ 495 56 check_source_sys_admin(rcp); 496 57 if (0 != rcp->rc_backup_app) 497 58 { 498 59 rc = 599 60 } </pre>	<pre> 501 1 502 2 } 503 3 else 504 4 { 505 5 /* For network backups, init the workitem & mark context: */ 506 6 rc = RSTL_SetWorkItem(rcp, thname); 507 7 if (rc != E_SUCCESS) 508 8 rc = allow_plane_arrays(rcp); 509 9 } 510 10 if (rc != E_SUCCESS) 511 11 { 512 12 free(rcp->rc_top_level_object_name); 513 13 rcp->rc_top_level_object_name = NULL; 514 14 return(rc); 515 15 } 516 16 if (0 != rcp->rc_backup_app) 517 17 { 518 18 return rcp -> currentPiptr -> pipFuncArray[pFuncIndexOfENO] 519 19 (rcp, 520 20 tloptr, 521 21 RSTRPC_tlo_type, 522 22 cookie, 523 23 maxbytes, 524 24 cntbytes, 525 25 allowbf); 526 26 } 527 27 else 528 28 { 529 29 /* for network backups retrieve content for the root directory. */ 530 30 return RSTL_GetDirContents(rcp, 531 31 /*, 532 32 allowbf, 533 33 maxbytes, 534 34 cntbytes, 535 35 cookie); 536 36 } 537 37 } 538 38 539 39 /* GetTLContents */ 540 40 541 41 } </pre>				
File Jan 04 16:35:25 2008	RSLgetrcb.c 9	Page 21 of 248	File Jan 04 16:35:25 2008	RSLgetrcb.c 10	Page 22 of 248


```

2  /*****
3  **
4  ** File Name: RLLfuncs.c
5  **
6  ** Copyright (c) 1998, 1999 by EMC Corporation.
7  **
8  ** Purpose:
9  ** This module contains the Restore Service Library's legacy
10 ** function definitions.
11 **
12 **
13 ** Table of Contents:
14 ** -----
15 **
16 ** RSTLL_GetDirContents
17 ** RSTLL_SetWorkItem
18 ** RSTLL_FillRestObj
19 ** RSTLL_AddVolumeToList
20 ** RSTLL_RemoveVolumeFromList
21 ** RSTLL_FreeValidList
22 **
23 ** Internal Functions:
24 **
25 ** Chdir
26 ** ConcatFullName
27 **
28 **
29 ** Compile-Time Options:
30 ** This section must list any compile time definitions
31 ** which will affect this header.
32 **
33 *****/
34
35 /* The following provides an RCS id in the binary that can be located
36 ** with the what(1) utility. The intent is to keep this short.
37 **
38 *****/
39 #ifndef lint
40 static char RCS_id[] = "RCS:11165 *
41 **Revisions *
42 **
43 #endif
44
45 /*
46 ** Feature test switches.
47 ** Standard defines required to turn on OS features go here.
48 **
49 ** The following is required for code that uses POSIX APIs.
50 ** Remove for non-POSIX, non-portable code.
51 **
52 **
53 #define _POSIX_SOURCE 1
54
55 /*
56 ** System headers.
57 **
58 */
59
60 /*
61 ** Bunch headers.
62 **
63 *****/
64
65 .lib/rstool/RLLfuncs.c 1
66
67 Page 25 of 248

```

```

64 #include <cb/ab/port.h>
65 #include <cb/rb/log.h>
66 #include <cbutil/ab/locking.h>
67 #include <cbm/libdrv.h>
68
69
70
71 /*
72 ** Local headers
73 **
74 #include <ebconfig/rbconfig.h>
75
76 #include <restore/RSLlegacy.h>
77
78
79
80 /*
81 ** #defined, structures, typedefs local to this source file
82 **
83
84
85 /*
86 ** External declarations
87 **
88
89 NEW_SRC_FILE()
90
91
92 /*
93 ** Local function prototypes
94 **
95
96 static eerrno_ty Chdir( restore_context *rcpt, char *filename );
97 /* FindNode is not being used, but left in place for possible
98 ** future use.
99 static eerrno_ty FindNode( tree_node *nptr, char **fname );
100
101
102 static eerrno_ty ConcatFullName( restore_context *rcpt,
103 tree_node *nptr,
104 char **fname );
105
106
107 /*
108 ** Function: RSTLL_SetWorkItem()
109 **
110 ** Function Description:
111 ** This function sets up the necessary fields in the
112 ** structure according to the specified work item.
113 **
114 ** Parameters:
115 ** rcpt (I) pointer to restore context
116 ** wItem (I) ptr to name of the work item.
117 **
118 ** Return Codes:
119 ** E_SUCCESS - success
120 ** E_RR_RECOVER_INVALID - Invalid work item
121 ** E_RR_RECOVER_MCAP_ERR - rc_remapcat() all failed
122
123 Note:
124 The rc_pwd field in recover_context is set to "/" in
125 rc_remapcat().
126
127
128 eerrno_ty
129 RSTLL_SetWorkItem( restore_context *rcpt, char *wItem )
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000

```

Page 27 of 248	RSSTL_SetWorkItem	Fri Jan 04 16:35:25 2008
128 1 129 1 130 1 131 1 132 1 133 1 134 1 135 1 136 1 137 1 138 2 139 2 140 2 141 2 142 2 143 2 144 2 145 1 146 1 147 1 148 1 149 1 150 1 151 1 152 1 153 1 154 2 155 2 156 1 157 1 158 1 159 1 160 1 161 1 162 1 163 1 164 1 165 1 166 1 167 2 168 2 169 2 170 2 171 2 172 2 173 3 174 3 175 3 176 3 177 3 178 3 179 2 180 2 181 2 182 2 183 2 184 2 185 3 186 3 187 3 188 2 189 2 190 2 191 2 192 1 193 2 194 2 195 2 196 2 197 2 198 2 199 2 200 2 201 2 202 2 203 2 204 2 205 3 206 3 207 2 208 2 209 3 210 3 211 2 212 2 213 2 214 3 215 3 216 2 217 2 218 2 219 2 220 3 221 3 222 3 223 2 224 2 225 1 226 1 227 1 228 1	<pre> errnum_t errnum; /* * Clear fields in recover context that are related to * the previously selected workitem/backup. * Note that when rc.candidate is zero, it means that there * is not a valid current backup. */ if (rcp->rc_have_wllock) /* Free previous workitem lock if any */ { rthe_log_debug(0, "RSSTL_SetWorkItem: Unlocking %s", rcp->rc_workitem_name); eh_unlock_object(RB_OBJECT_WORKITEM, rcp->rc_workitem_name, rcp->rc_have_wllock = 0); } rcp->rc_candidate = (time_t)0; memset(rcp->rc_candidate_ptr, 0, CANDIDATE_STRING_SIZE); /* Set the current workitem name: */ rcp->rc_workitem_name = eol_strdup(wNameP); if (NULL == rcp->rc_workitem_name) { rcg_apl_log_err(SUB_CSN_KNOWN, NULL); return(EP_RB_RECOVER_NOMEM); } /* * Reset the multiple structures for this work item. * rc.memorct() will lock the work item indirectly * through the routines it calls(). */ errnum = rc.memorct(rcp); /* set up the meat for the work item */ if (0 != errnum) { /* * If rc.memorct() failed, we must unlock the work item * and reset the env. */ if (rcp->rc_have_wllock) { rthe_log_debug(0, "RSSTL_SetWorkItem: Unlocking %s", rcp->rc_workitem_name); eh_unlock_object(RB_OBJECT_WORKITEM, rcp->rc_workitem_name, RB_UNLOCK_FREE_IT); } rcp->rc_have_wllock = 0; } free(rcp->rc_workitem_name); rcp->rc_workitem_name = NULL; /* I don't think we want to do this for all apps */ if (NULL != rcp->rc_template_name) { free(rcp->rc_template_name); rcp->rc_template_name = NULL; } rcp->rc_saveset_thread = EB_THREAD_PRIORITY; #endif if (</pre>	<pre> EP_RB_CSN_NO_SAVEDSET != errnum && EP_RB_RECOVER_NO_SAVEDSET != errnum) { rthe_internal_error(errnum, "rc.memorct failed"); return(errnum); } if (NULL == rcp->rc_template_name) { cat_descriptor "catBptr: rthead head t catBptr: rthesetset_c_saveset; catBptr = mem.getcatd(rcp->rc_jcpd, mem.get_jcpplane(rcp->rc_jcpd)); if (NULL == catBptr) { return(EP_RB_RECOVER_NO_CATALOG); } if (0 != cat.get_cat_head(catBptr, kcalhr)) { return(EP_RB_RECOVER_NO_CATALOG); } if (0 != ss.find1(catBptr.ch_savesetID, ssaveset, 0)) { return(EP_RB_RECOVER_NO_SAVEDSET); } rcp->rc_template_name = eol_strdup(ssaveset.ss_temp1); if (NULL == rcp->rc_template_name) { rcg_apl_log_err(SUB_CSN_KNOWN, NULL); return(EP_RB_RECOVER_NOMEM); } rcp->rc_saveset_thread = saveset.ss_thread; } return(EB_SUCCESS); /* RSSTL_SetWorkItem */ </pre>
Page 28 of 248	RSSTL_SetWorkItem	Fri Jan 04 16:35:25 2008

Page 27 of 248	RSSTL_SetWorkItem	Fri Jan 04 16:35:25 2008
Page 27 of 248	./libe_restoreFillUnsc.c 3	Fri Jan 04 16:35:25 2008
Page 28 of 248	RSSTL_SetWorkItem	Fri Jan 04 16:35:25 2008
Page 28 of 248	./libe_restoreFillUnsc.c 4	Fri Jan 04 16:35:25 2008

```

230 /*
231  * Function: RSTL_GetDirContents()
232  *
233  * Function Description:
234  *   This function returns the contents of the current directory
235  *   in the
236  *   preallocated rescanobj object buffers.
237  *
238  * Parameters:
239  *   tcp - (I) Pointer to the restore context
240  *   dirName - (I) ptr to the directory full pathname string
241  *   allowMBF - (I) flag for whether or not to include bad files
242  *   maxEntries - (I) max. # of entries to return
243  *   objListPtr - (O) a pointer to receive the start of the object list
244  *   cntEntries - (O) ptr to buffer to receive number of entries returned
245  *   cookie - (O) ptr to a token used in successive calls when
246  *             more objects remain.
247  *
248  *   The input value must be
249  *   INTL_COOKIE on the first call for a work
250  *   will be DONE_COOKIE on return if the object
251  *   a workitem is returned;
252  *   otherwise it is meaningful
253  *   only to the internals of the API.
254  *
255  * Return Codes:
256  *   E_SUCCESS - success
257  *   E_R_BROKEN - input argument
258  *   EP_RB_RECOVER_BAD_ARGS - cannot find tree node for curr dir
259  *   EP_RB_RECOVER_INVALIDDIR - cannot find tree node for curr dir
260  *   EP_RB_RECOVER_BAD_COOKIE - invalid cookie
261  *   EP_RB_RECOVER_INVALIDOP - invalid operation
262  *   EP_RB_RECOVER_FATALERR - internal inconsistency
263  */
264
265 RSTL_GetDirContents( trestore_context
266                    char
267                    const boolean_t
268                    const long
269                    struct RSTRFC_ufo_list
270                    long
271                    int
272                    {
273 1. tcp = NULL;
274 2. dir_name = NULL;
275 3. allowMBF = **trp;
276 4. static long valid_cookie = INTL_COOKIE;
277 5. static long valid_cookie = INTL_COOKIE;
278 6. struct RSTRFC_ufo_list *rescanobj = NULL;
279 7. struct RSTRFC_ufo_list
280 8. int output_count;
281 9. trestore_ctx_t ctx;
282 10.
283 11. if (objListPtr == NULL)
284 12. {
285 13. return(EP_RB_RECOVER_BAD_ARGS);
286 14. }

```

```

287 18. /*
288 19.  * If it's the first call on the specified directory, we must:
289 20.  * 1. do a child operation to get the recover_context environment
290 21.  *   set up
291 22.  *   appropriately;
292 23.  * 2. free the dir_ale array and reset tree_node count nItems;
293 24.  * 3. get all the tree nodes for the directory contents in dir_ale
294 25.  */
295
296 26. if (*cookie == INTL_COOKIE)
297 27. {
298 28. if (trc = CmbDir(tcp, dirName) != E_SUCCESS)
299 29. {
300 30. return(trc);
301 31. }
302 32. if (NULL != dir_ale)
303 33. {
304 34. mcat_array_list(tcp->rc_mcp, dir_ale);
305 35. dir_ale = NULL;
306 36. }
307 37.
308 38. nItems = 0;
309 39. valid_cookie = INTL_COOKIE;
310 40.
311 41. /*
312 42.  * Get the tree node for the current dir, which is needed for
313 43.  * the next mcat call.
314 44.  */
315 45.
316 46. if ((trp = mcat_get_current_dir(tcp->rc_mcp)) == (
317 47. {
318 48. return(EP_RB_RECOVER_INVALIDDIR);
319 49. }
320 50. nItems = mcat_array_children(tcp->rc_mcp, trp, &dir_ale);
321 51. trp = dir_ale;
322 52.
323 53. }
324 54. else
325 55. {
326 56. /*
327 57.  * We're doing a follow-up query on a directory.
328 58.  * value different from where it was left off.
329 59.  * If the cookie has
330 60.  * is invalid and an error code is returned. If the specified
331 61.  * dirName is different from what's set previously.
332 62.  * Then this is an
333 63.  * invalid query.
334 64.  * Otherwise set the tree_node ptr to what's in the
335 65.  * cookie.
336 66.  */
337 67. if (*cookie != valid_cookie)
338 68. {
339 69. return(EP_RB_RECOVER_BAD_COOKIE);
340 70. }
341 71. if (0 != strcmp(dirName, tcp->rc_pwd))
342 72. {
343 73. return(EP_RB_RECOVER_INVALIDOP);
344 74. }
345 75. trp = (tree_node **) cookie;
346 76. }

```

```

348 1      if (ntItems == 0)
349 2      {
350 3          *contentPtr = 0;
351 4          *cookie = DONE_COOKIE;
352 5          valid_cookie = DONE_COOKIE;
353 6      }
354 7      /*
355 8       * Release the memory when we're done
356 9       */
357 10     if (NULL != dir_e1e)
358 11     {
359 12         mcat_array_rls(tcp->rc_mcp, dir_e1e);
360 13         dir_e1e = NULL;
361 14     }
362 15     return(E_SUCCESS);
363 16 }
364 17
365 18 output_count = 0;
366 19 while ((ntItems > 0) && (output_count < maxEntries))
367 20 {
368 21     if (*objListPtr == NULL) {
369 22         *objListPtr = calloc(1, sizeof(
370 23             struct RSTRPC_uro_list));
371 24     }
372 25     else {
373 26         listPtr->next = calloc(1, sizeof(
374 27             struct RSTRPC_uro_list));
375 28     }
376 29     listPtr = listPtr->next;
377 30
378 31     if (NULL == listPtr) { /* test for malloc failure */
379 32         rtc = EP_RB_RECOVER_NOMEM;
380 33         break;
381 34     }
382 35     objPtr = calloc(1, sizeof(
383 36         struct RSTRPC_user_restorable_object));
384 37     if (! (listPtr->uro = objPtr) ) {
385 38         rtc = EP_RB_RECOVER_NOMEM;
386 39         break;
387 40     }
388 41     if (*tmp == NULL)
389 42     {
390 43         rtc = EP_RB_RECOVER_FATALERR;
391 44         break;
392 45     }
393 46
394 47     /*
395 48      * get the catalog element
396 49      */
397 50     mcat_getcatIn(tcp->rc_mcp, tmp, &cat_elem);
398 51
399 52     /*
400 53      * Check if the caller wants to include bad files and
401 54      * the status of the current object in these areas
402 55      */
403 56     if (ALLOWBAD && (cat_elem.ce_status & CSTAT_BADDATA))
404 57     {
405 58         tmp++;
406 59         ntItems--;
407 60         continue;
408 61     }
409 62 }

```

```

410 2    }
411 3    /* fill output buffers */
412 4    rtc = RSTLL_FillRestObj(tcp, &cat_elem, tmp, objPtr);
413 5    if (
414 6        rtc != 0) /* if something went wrong in RSTLL_FillRestObj() */
415 7    {
416 8        break; /* we must stop processing further */
417 9    }
418 10    output_count++;
419 11    tmp++;
420 12    ntItems--;
421 13 }
422 14
423 15 *contentPtr = output_count;
424 16
425 17 /*
426 18  * If we exited the loop with the preallo'd buffers filled,
427 19  * we may not have exhausted the dir contents
428 20  */
429 21 if (output_count == maxEntries)
430 22 {
431 23     if (
432 24         ntItems == 0) /* dir contents exhausted,
433 25         */
434 26     {
435 27         *cookie = DONE_COOKIE; /* set the cookie accordingly
436 28         */
437 29     }
438 30     else /* otherwise,
439 31         store the tree
440 32         */
441 33     {
442 34         *cookie = (long) tmp; /* node ptr in the cookie
443 35         */
444 36     }
445 37 }
446 38
447 39 /*cookie = DONE_COOKIE;
448 40
449 41 */
450 42
451 43 /*
452 44  * Release the memory when we're done
453 45  */
454 46 if (*cookie == DONE_COOKIE)
455 47 {
456 48     if (NULL != dir_e1e)
457 49     {
458 50         mcat_array_rls(tcp->rc_mcp, dir_e1e);
459 51         dir_e1e = NULL;
460 52     }
461 53     valid_cookie = DONE_COOKIE;
462 54 }
463 55 else
464 56 {
465 57     valid_cookie = *cookie;
466 58 }
467 59
468 60 if (rtc != 0)

```

```

470 2      (
471 2          /* Free memory allocated here */
472 2          RSTLL_FreeRestorableObjList( objListPtr );
473 2          /* leave NULL in *objListPtr */
474 2          return(rtc);
475 2      )
476 1      else
477 1      {
478 2          return(E_SUCCESS);
479 2      }
480 1      /* RSTLL_GmDirContents */
481  }

```

```

483 2      /*
484 2      * Function: RSTLL_FillRstObj()
485 2      *
486 2      * Function Description:
487 2      *   This function fills most of the fields of the
488 2      *   a file or directory object using info contained in the input
489 2      *   element structure and tree node structure.
490 2      *
491 2      * Parameters:
492 2      *   rcpt
493 2      *   catElemPtr
494 2      *   tnpPtr
495 2      *   objPtr
496 2      *   0) ptr to the RSTRRC user_restorable_object
497 2      *   to be filled
498 2      *
499 2      * Return Codes:
500 2      *   E_SUCCESS
501 2      *   EP_RB_RECOVER_FATALERR - operation is successful;
502 2      *   EP_RB_RECOVER_FATALERR - inconsistency exists in the recover
503 2      *   EP_RB_RECOVER_WARNERR - internal, caller should abort;
504 2      *   EP_RB_RECOVER_WARNERR - not enough memory
505 2      *
506 2      * Note: The portion of code that calculates the size of the directory
507 2      *   is copied from sizing() in src/server/libb/obrcvover/
508 2      *   grandfathered/cmd_is.c
509 2      */
510 2      return(E_SUCCESS);
511 2      /* RSTLL_FillRstObj() restore context *rcpt,
512 2      *   tnode, elem_t *catElemPtr,
513 2      *   struct RSTRRC_user_restorable_object *objPtr)
514 2      {
515 2      /*
516 2      * The rtree_elem_t passed in as the first arg does not contain
517 2      * the object's full pathname. This is because most of the
518 2      * fields of the rtree_elem_t are filled in by the RSTRRC buffers()
519 2      * function. The only field that is not filled in is the pathname.
520 2      * Call Clnm2tree() which will fill the pathname if everything
521 2      * in the catalog was correct.
522 2      */
523 2      char *fname;
524 2      char *tmp_uid;
525 2      char *tmp_gid;
526 2      char *tmp_perms;
527 2      char *tmp_mode;
528 2      int nochildren;
529 2      u_long phony_size;
530 2      rtree_elem_t tree_elem;
531 2      int errnum;
532 2      return(E_SUCCESS);
533 2      /* validate inputs: */
534 2      if (NULL == rcpt || NULL == catElemPtr || NULL == tnpPtr || NULL ==
535 2      objPtr)
536 2      {
537 2          return(EP_RB_RECOVER_FATALERR);
538 2      }
539 2      return(E_SUCCESS);
540 2      }

```

542 1	/* alloc and clear net backup specific (opaque) structure */
543 1	appdata_ptr = calloc(1, sizeof(netBackupObjData));
544 1	if (NULL == appdata_ptr)
545 2	{
546 2	rec_api_log_csm(SUB_CSM_NOMEX, NULL);
547 2	return(EP_RB_RECOVER_NOMEX);
548 1	obj_ptr->appdata_len = sizeof(netBackupObjData);
549 1	obj_ptr->appdata_data = (char *)appdata_ptr;
550 1	return(0);
552 1	obj_ptr->root_backupApp = rcp->rc_backupApp; /* might not be network */
554 1	errnum = climo2recs(tcp,
555 1	tnptr->tn_index,
556 1	tnptr->tn_name,
557 1	&errnum,
558 1	&tnptr);
559 1	if (errnum != 0) { ifname == NULL; }
561 1	{
562 2	/*
563 2	When climo2recs() returns an error and a NULL name ptr,
564 2	it's most likely, that we're dealing with the case that
565 2	the object has been removed from the file system.
566 2	Therefore we cannot find the node's full pathname. This
567 2	happens when the work item has filesize such as "00 DIR
568 2	/XYZ/". ConcatFullName() concatenates the full pathname
569 2	of the object by traversing the tree node chain upwards.
570 2	*/
571 2	ifc = ConcatFullName(tcp, tnptr, &iname);
572 2	if (E_SOCKSS != ifc)
573 1	{
574 1	return(ifc);
575 1	}
576 1	/*
577 1	ConcatFullName has malloc'd the fullname string buffer,
578 1	so we don't need to reallocate it.
579 1	*/
580 1	obj_ptr->root_objName = ifname;
581 1	}
582 1	else
583 1	{
584 1	obj_ptr->root_objName = esl_strdup(ifname);
585 1	}
586 1	if (NULL == obj_ptr->root_objName)
587 1	{
588 1	rec_api_log_csm(SUB_CSM_NOMEX, NULL);
589 1	return(EP_RB_RECOVER_NOMEX);
590 1	}
591 1	appdata_ptr->objSSID = catEmblptr->oe_ssid;
592 1	appdata_ptr->objPID = catEmblptr->oe_bttileid;
593 1	obj_ptr->objMode = catEmblptr->oe_mode;
594 1	if (((NULL == tmp_uuid = uuidStr((int)catEmblptr->oe_owner)))
595 1	(NULL == tmp_uuid = guidStr((int)catEmblptr->oe_group))))
596 1	return EP_RB_RECOVER_NOMEX;

608 1	obj_ptr->objOwnerName = esl_strdup(tmp_uuid);
609 1	if (NULL == obj_ptr->objOwnerName)
610 2	{
611 2	rec_api_log_csm(SUB_CSM_NOMEX, NULL);
612 2	return(EP_RB_RECOVER_NOMEX);
613 1	}
614 1	obj_ptr->objGroupName = esl_strdup(tmp_gid);
615 1	if (NULL == obj_ptr->objGroupName)
616 1	{
617 2	rec_api_log_csm(SUB_CSM_NOMEX, NULL);
618 2	return(EP_RB_RECOVER_NOMEX);
619 1	}
620 1	obj_ptr->objModTime = catEmblptr->oe_mtime;
621 1	if (catEmblptr->oe_status & CSMPTL_DATAINT)
622 1	{
623 2	obj_ptr->objBackupStatus = RSTRPC_Backup_Bad;
624 2	}
625 1	obj_ptr->objBackupStatus = RSTRPC_Backup_Bad;
626 1	else
627 1	{
628 2	obj_ptr->objBackupStatus = RSTRPC_Backup_Good;
629 2	appdata_ptr->objInCName = tnptr->tn_name;
630 1	appdata_ptr->objInIndex = tnptr->tn_index;
631 1	appdata_ptr->objInIndex = tnptr->tn_index;
632 1	if ((NULL == tnptr->tn_name) && (tnptr->tn_flags & TWF_ROOTDIR))
633 1	{
634 2	obj_ptr->objBaseName = esl_strdup("");
635 2	}
636 1	else
637 1	{
638 2	obj_ptr->objBaseName = esl_strdup(tnptr->tn_name);
639 2	}
640 1	if (NULL == obj_ptr->objBaseName)
641 1	{
642 2	rec_api_log_csm(SUB_CSM_NOMEX, NULL);
643 2	return(EP_RB_RECOVER_NOMEX);
644 1	}
645 1	/*
646 1	Calculation of the size of the object depends on the type of
647 1	the
648 1	* object as follows:
649 1	* If the object is a directory, we estimate its size;
650 1	* If the object is a file, we get its true size in bytes;
651 1	* If the object is a block or char device node,
652 1	* we treat
653 1	* it as a file but with size 0.
654 1	*/
655 1	if (S_ISDIR(catEmblptr->oe_mode))
656 1	{
657 1	/*
658 1	Directory sizes are not stored in the catalogs,
659 1	we construct a phony size based on the number
660 1	of children in the directory, just so that
661 1	the size in the is listing looks "comforting",
662 1	and to help suggest which directories have lots
663 1	of files vs. few files.
664 1	*/
665 1	obj_ptr->root_objLevel = RSTRPC_containsType;
666 1	}
667 1	}
668 1	}

```

671 2      /*
672 2      */
673 2      *do this just to get the number of children
674 2      */
675 2      nchildren = mcat_array.children(
676 2      mcat_array.flags > rc_mcp, tnptr, &n_children);
677 2      /*
678 2      * derive phony size @ 20bytes/entry, rounded to 512 mult
679 2      */
680 2      phony_size = nchildren * 20;
681 2      phony_size = (phony_size + 511) / 512;
682 2      phony_size = phony_size * 512;
683 2      if (phony_size == 0)
684 2      {
685 2      phony_size = 512;
686 2      }
687 2      objPtr->objSize.low = phony_size;
688 2      objPtr->objSize.high = 0;
689 2      else
690 2      {
691 2      objPtr->root.objLevel = RSTRRC_leaf_Type;
692 2      /*
693 2      * For char or block device node files, the size is used to
694 2      * indicate its devno.
695 2      */
696 2      if (S_ISCHR(catElemPtr->ce_mode) || S_ISBLK(
697 2      catElemPtr->ce_mode))
698 2      {
699 2      objPtr->objSize.low = catElemPtr->ce_dev;
700 2      objPtr->objSize.high = 0;
701 2      }
702 2      else
703 2      {
704 2      objPtr->objSize.high = catElemPtr->ce_filesSize.high;
705 2      objPtr->objSize.low = catElemPtr->ce_filesSize.low;
706 2      }
707 2      return(E_SUCCESS);
708 2      /* RSTL_fillReach */
709 2      }
710 2      }
711 2      }
712 2      }
713 2      }
714 2      }
715 2      }

```

```

717 2      /*
718 2      * Function: ChDir()
719 2      *
720 2      * Function Description:
721 2      * This function changes the current directory to the specified
722 2      * All the fields in the recover_context structure related to
723 2      * directory get reset appropriately.
724 2      *
725 2      * Parameters:
726 2      * (1) Pointer to the restore context
727 2      *
728 2      * dirName
729 2      * I/P to the directory full pathname NULL-terminated string
730 2      *
731 2      * Return Codes:
732 2      * E_SUCCESS
733 2      * EP_RB_RECOVER_INVALIDDIR - invalid directory
734 2      * EP_RB_RECOVER_PERMISSION_DENIED - caller has no
735 2      * access rights
736 2      *
737 2      * Note: This function was copied from the original xbsrecover.c
738 2      * base_sys.c:
739 2      * rb_change_dir(), which is an existing working piece, with minor
740 2      * modifications. To reduce risk, the original program logic is
741 2      * preserved.
742 2      */
743 2      static void by
744 2      ChDir(Restore_Context *rcptr, char *dirName)
745 2      {
746 2      tree_node
747 2      int
748 2      err = 0;
749 2      char
750 2      *save_current_pwd;
751 2      save_current_pwd = rcptr->rc_pwd; /* May be freed later */
752 2      ntmp = mcat_lookup_path_chk(rcptr->rc_mcp,
753 2      dirName,
754 2      rcptr->rc_effective_uid,
755 2      rcptr->rc_effective_gid,
756 2      rcptr->rc_effective_ruids,
757 2      rcptr->rc_effective_gids,
758 2      rcptr->rc_effective_ruids,
759 2      rcptr->rc_effective_gids,
760 2      rcptr->rc_effective_ruids,
761 2      rcptr->rc_effective_gids,
762 2      rcptr->rc_effective_ruids,
763 2      rcptr->rc_effective_gids,
764 2      rcptr->rc_effective_ruids,
765 2      rcptr->rc_effective_gids,
766 2      rcptr->rc_effective_ruids,
767 2      rcptr->rc_effective_gids,
768 2      rcptr->rc_effective_ruids,
769 2      rcptr->rc_effective_gids,
770 2      rcptr->rc_effective_ruids,
771 2      rcptr->rc_effective_gids,
772 2      rcptr->rc_effective_ruids,
773 2      rcptr->rc_effective_gids,
774 2      rcptr->rc_effective_ruids,
775 2      rcptr->rc_effective_gids,
776 2      rcptr->rc_effective_ruids,
777 2      rcptr->rc_effective_gids,
778 2      rcptr->rc_effective_ruids,
779 2      rcptr->rc_effective_gids,
780 2      rcptr->rc_effective_ruids,
781 2      rcptr->rc_effective_gids,
782 2      rcptr->rc_effective_ruids,
783 2      rcptr->rc_effective_gids,
784 2      rcptr->rc_effective_ruids,
785 2      rcptr->rc_effective_gids,
786 2      rcptr->rc_effective_ruids,
787 2      rcptr->rc_effective_gids,
788 2      rcptr->rc_effective_ruids,
789 2      rcptr->rc_effective_gids,
790 2      rcptr->rc_effective_ruids,
791 2      rcptr->rc_effective_gids,
792 2      rcptr->rc_effective_ruids,
793 2      rcptr->rc_effective_gids,
794 2      rcptr->rc_effective_ruids,
795 2      rcptr->rc_effective_gids,
796 2      rcptr->rc_effective_ruids,
797 2      rcptr->rc_effective_gids,
798 2      rcptr->rc_effective_ruids,
799 2      rcptr->rc_effective_gids,
800 2      rcptr->rc_effective_ruids,
801 2      rcptr->rc_effective_gids,
802 2      rcptr->rc_effective_ruids,
803 2      rcptr->rc_effective_gids,
804 2      rcptr->rc_effective_ruids,
805 2      rcptr->rc_effective_gids,
806 2      rcptr->rc_effective_ruids,
807 2      rcptr->rc_effective_gids,
808 2      rcptr->rc_effective_ruids,
809 2      rcptr->rc_effective_gids,
810 2      rcptr->rc_effective_ruids,
811 2      rcptr->rc_effective_gids,
812 2      rcptr->rc_effective_ruids,
813 2      rcptr->rc_effective_gids,
814 2      rcptr->rc_effective_ruids,
815 2      rcptr->rc_effective_gids,
816 2      rcptr->rc_effective_ruids,
817 2      rcptr->rc_effective_gids,
818 2      rcptr->rc_effective_ruids,
819 2      rcptr->rc_effective_gids,
820 2      rcptr->rc_effective_ruids,
821 2      rcptr->rc_effective_gids,
822 2      rcptr->rc_effective_ruids,
823 2      rcptr->rc_effective_gids,
824 2      rcptr->rc_effective_ruids,
825 2      rcptr->rc_effective_gids,
826 2      rcptr->rc_effective_ruids,
827 2      rcptr->rc_effective_gids,
828 2      rcptr->rc_effective_ruids,
829 2      rcptr->rc_effective_gids,
830 2      rcptr->rc_effective_ruids,
831 2      rcptr->rc_effective_gids,
832 2      rcptr->rc_effective_ruids,
833 2      rcptr->rc_effective_gids,
834 2      rcptr->rc_effective_ruids,
835 2      rcptr->rc_effective_gids,
836 2      rcptr->rc_effective_ruids,
837 2      rcptr->rc_effective_gids,
838 2      rcptr->rc_effective_ruids,
839 2      rcptr->rc_effective_gids,
840 2      rcptr->rc_effective_ruids,
841 2      rcptr->rc_effective_gids,
842 2      rcptr->rc_effective_ruids,
843 2      rcptr->rc_effective_gids,
844 2      rcptr->rc_effective_ruids,
845 2      rcptr->rc_effective_gids,
846 2      rcptr->rc_effective_ruids,
847 2      rcptr->rc_effective_gids,
848 2      rcptr->rc_effective_ruids,
849 2      rcptr->rc_effective_gids,
850 2      rcptr->rc_effective_ruids,
851 2      rcptr->rc_effective_gids,
852 2      rcptr->rc_effective_ruids,
853 2      rcptr->rc_effective_gids,
854 2      rcptr->rc_effective_ruids,
855 2      rcptr->rc_effective_gids,
856 2      rcptr->rc_effective_ruids,
857 2      rcptr->rc_effective_gids,
858 2      rcptr->rc_effective_ruids,
859 2      rcptr->rc_effective_gids,
860 2      rcptr->rc_effective_ruids,
861 2      rcptr->rc_effective_gids,
862 2      rcptr->rc_effective_ruids,
863 2      rcptr->rc_effective_gids,
864 2      rcptr->rc_effective_ruids,
865 2      rcptr->rc_effective_gids,
866 2      rcptr->rc_effective_ruids,
867 2      rcptr->rc_effective_gids,
868 2      rcptr->rc_effective_ruids,
869 2      rcptr->rc_effective_gids,
870 2      rcptr->rc_effective_ruids,
871 2      rcptr->rc_effective_gids,
872 2      rcptr->rc_effective_ruids,
873 2      rcptr->rc_effective_gids,
874 2      rcptr->rc_effective_ruids,
875 2      rcptr->rc_effective_gids,
876 2      rcptr->rc_effective_ruids,
877 2      rcptr->rc_effective_gids,
878 2      rcptr->rc_effective_ruids,
879 2      rcptr->rc_effective_gids,
880 2      rcptr->rc_effective_ruids,
881 2      rcptr->rc_effective_gids,
882 2      rcptr->rc_effective_ruids,
883 2      rcptr->rc_effective_gids,
884 2      rcptr->rc_effective_ruids,
885 2      rcptr->rc_effective_gids,
886 2      rcptr->rc_effective_ruids,
887 2      rcptr->rc_effective_gids,
888 2      rcptr->rc_effective_ruids,
889 2      rcptr->rc_effective_gids,
890 2      rcptr->rc_effective_ruids,
891 2      rcptr->rc_effective_gids,
892 2      rcptr->rc_effective_ruids,
893 2      rcptr->rc_effective_gids,
894 2      rcptr->rc_effective_ruids,
895 2      rcptr->rc_effective_gids,
896 2      rcptr->rc_effective_ruids,
897 2      rcptr->rc_effective_gids,
898 2      rcptr->rc_effective_ruids,
899 2      rcptr->rc_effective_gids,
900 2      rcptr->rc_effective_ruids,
901 2      rcptr->rc_effective_gids,
902 2      rcptr->rc_effective_ruids,
903 2      rcptr->rc_effective_gids,
904 2      rcptr->rc_effective_ruids,
905 2      rcptr->rc_effective_gids,
906 2      rcptr->rc_effective_ruids,
907 2      rcptr->rc_effective_gids,
908 2      rcptr->rc_effective_ruids,
909 2      rcptr->rc_effective_gids,
910 2      rcptr->rc_effective_ruids,
911 2      rcptr->rc_effective_gids,
912 2      rcptr->rc_effective_ruids,
913 2      rcptr->rc_effective_gids,
914 2      rcptr->rc_effective_ruids,
915 2      rcptr->rc_effective_gids,
916 2      rcptr->rc_effective_ruids,
917 2      rcptr->rc_effective_gids,
918 2      rcptr->rc_effective_ruids,
919 2      rcptr->rc_effective_gids,
920 2      rcptr->rc_effective_ruids,
921 2      rcptr->rc_effective_gids,
922 2      rcptr->rc_effective_ruids,
923 2      rcptr->rc_effective_gids,
924 2      rcptr->rc_effective_ruids,
925 2      rcptr->rc_effective_gids,
926 2      rcptr->rc_effective_ruids,
927 2      rcptr->rc_effective_gids,
928 2      rcptr->rc_effective_ruids,
929 2      rcptr->rc_effective_gids,
930 2      rcptr->rc_effective_ruids,
931 2      rcptr->rc_effective_gids,
932 2      rcptr->rc_effective_ruids,
933 2      rcptr->rc_effective_gids,
934 2      rcptr->rc_effective_ruids,
935 2      rcptr->rc_effective_gids,
936 2      rcptr->rc_effective_ruids,
937 2      rcptr->rc_effective_gids,
938 2      rcptr->rc_effective_ruids,
939 2      rcptr->rc_effective_gids,
940 2      rcptr->rc_effective_ruids,
941 2      rcptr->rc_effective_gids,
942 2      rcptr->rc_effective_ruids,
943 2      rcptr->rc_effective_gids,
944 2      rcptr->rc_effective_ruids,
945 2      rcptr->rc_effective_gids,
946 2      rcptr->rc_effective_ruids,
947 2      rcptr->rc_effective_gids,
948 2      rcptr->rc_effective_ruids,
949 2      rcptr->rc_effective_gids,
950 2      rcptr->rc_effective_ruids,
951 2      rcptr->rc_effective_gids,
952 2      rcptr->rc_effective_ruids,
953 2      rcptr->rc_effective_gids,
954 2      rcptr->rc_effective_ruids,
955 2      rcptr->rc_effective_gids,
956 2      rcptr->rc_effective_ruids,
957 2      rcptr->rc_effective_gids,
958 2      rcptr->rc_effective_ruids,
959 2      rcptr->rc_effective_gids,
960 2      rcptr->rc_effective_ruids,
961 2      rcptr->rc_effective_gids,
962 2      rcptr->rc_effective_ruids,
963 2      rcptr->rc_effective_gids,
964 2      rcptr->rc_effective_ruids,
965 2      rcptr->rc_effective_gids,
966 2      rcptr->rc_effective_ruids,
967 2      rcptr->rc_effective_gids,
968 2      rcptr->rc_effective_ruids,
969 2      rcptr->rc_effective_gids,
970 2      rcptr->rc_effective_ruids,
971 2      rcptr->rc_effective_gids,
972 2      rcptr->rc_effective_ruids,
973 2      rcptr->rc_effective_gids,
974 2      rcptr->rc_effective_ruids,
975 2      rcptr->rc_effective_gids,
976 2      rcptr->rc_effective_ruids,
977 2      rcptr->rc_effective_gids,
978 2      rcptr->rc_effective_ruids,
979 2      rcptr->rc_effective_gids,
980 2      rcptr->rc_effective_ruids,
981 2      rcptr->rc_effective_gids,
982 2      rcptr->rc_effective_ruids,
983 2      rcptr->rc_effective_gids,
984 2      rcptr->rc_effective_ruids,
985 2      rcptr->rc_effective_gids,
986 2      rcptr->rc_effective_ruids,
987 2      rcptr->rc_effective_gids,
988 2      rcptr->rc_effective_ruids,
989 2      rcptr->rc_effective_gids,
990 2      rcptr->rc_effective_ruids,
991 2      rcptr->rc_effective_gids,
992 2      rcptr->rc_effective_ruids,
993 2      rcptr->rc_effective_gids,
994 2      rcptr->rc_effective_ruids,
995 2      rcptr->rc_effective_gids,
996 2      rcptr->rc_effective_ruids,
997 2      rcptr->rc_effective_gids,
998 2      rcptr->rc_effective_ruids,
999 2      rcptr->rc_effective_gids,
1000 2      rcptr->rc_effective_ruids,

```



```

774 1 //
775 2 * Even though no error was returned, we have to apply the final
776 3 * permission check ourselves.
777 4 */
778 5
779 6 if (0 == err)
780 7 {
781 8     if (!check_permission(&rcptr, PERMCHK_R|PERMCHK_X, ntpd, (**)NULL))
782 9         local_item_t(**)NULL))
783 10 }
784 11 return(EP_RB_RECOVER_PERMISSION_DENIED);
785 12
786 13 if (NULL != dirName)
787 14 {
788 15     rcptr->rc_pwd = csal_strdup(dirName);
789 16     if (NULL == rcptr->rc_pwd)
790 17     {
791 18         free(save_current_pwd);
792 19         return(EP_RB_RECOVER_NOMEM);
793 20     }
794 21 }
795 22 return(EP_RB_RECOVER_INVALIDDIR);
796 23
797 24 }
798 25
799 26 }
800 27
801 28 }
802 29
803 30 }
804 31
805 32 }
806 33
807 34 }

```

```

810 1 //
811 2 * Function: ConcatFullName()
812 3 *
813 4 * Function Description:
814 5 * Given the ptr to a tree_node which is for a directory that
815 6 * does not have a valid corresponding catalog entry, this function
816 7 * concatenates its full pathname by traversing the tree_node
817 8 * upwards all the way to the root dir. It returns the ptr to
818 9 * the concatenated full pathname string in the provided string
819 10 * ptr.
820 11
821 12 Note that this function mallocs the buffer for the full
822 13 string. It is the caller's responsibility to free the memory
823 14 via the free() call.
824 15
825 16 * Parameters:
826 17 * rcptr - (I) Pointer to the restore context
827 18 * ntpd - (I) ptr to the tree_node
829 19 * fname - (O) ptr to a pathname string ptr
830 20
831 21 * Return Codes:
832 22 * E_SUCCESS - operation is successful;
833 23 * EP_RB_RECOVER_BAD_TREENODE - invalid treenode;
834 24 * EP_RB_RECOVER_NOMEM - malloc failed;
835 25 */

```

```

836 1 static void restore_name_by
837 2 ConcatFullName(restore_context *rcptr,
838 3 tree_node *treenode, char **fname)
839 4 {
840 5     tree_node *pntpr; /* parent tree_node ptr */
841 6     char *fntpr = NULL; /* ptr to string buffer of full pathname */
842 7     char *sptr = NULL;
843 8     size_t plen;
844 9     size_t c_len;
845 10
846 11
847 12
848 13
849 14
850 15
851 16
852 17
853 18
854 19
855 20
856 21
857 22
858 23
859 24
860 25
861 26
862 27
863 28
864 29
865 30
866 31
867 32
868 33
869 34
870 35
871 36

```

```

/*
 * Some basic validations.
 */

```

```

if (NULL == ntpd)
{
    return(EP_RB_RECOVER_BAD_TREENODE);
}

if (!(!ntpd->tn_flags & TNE_ISDIR))
{
    return(EP_RB_RECOVER_BAD_TREENODE);
}

if (NULL == ntpd->tn_name)
{
    return(EP_RB_RECOVER_BAD_TREENODE);
}

```

```

/*
 * If the ntpd points to the tree_node for the root dir,
 * then simply return "/" as the full pathname.
 */

```

```

872 1         if (tnPtr->tn_flags & TNP_ROOTDIR)
873 2         {
874 2             fmptr = malloc(2);
875 2             if (NULL == fmptr)
876 3             {
877 3                 rec_apl_log_cam(SUB_CSN_NOMEX, NULL);
878 3                 return(EP_RB_RECOVER_NOMEX);
879 2             }
880 2             fmptr[0] = '/';
881 2             fmptr[1] = 0;
882 2             *fname = fmptr;
883 2             return(E_SUCCESS);
884 1         }

886 1         clen = strlen(tnPtr->tn_name);
888 1         /*
889 1          * Alloc buffer for the node's name string (including the NULL
890 1          * terminator) plus a leading '/', then copy the current node's
891 1          * name into it with a leading '/'.
892 1          */
894 1         fmptr = malloc(clen + 2);
895 1         if (NULL == fmptr)
896 2         {
897 2             rec_apl_log_cam(SUB_CSN_NOMEX, NULL);
898 2             return(EP_RB_RECOVER_NOMEX);
899 1         }

901 1         fmptr[0] = '/';
902 1         memcpy(fmptr+1, tnPtr->tn_name, clen);
903 1         fmptr[1+clen] = 0;

905 1         ptnPtr = tnPtr->tn_parent;

907 1         /*
908 1          * Traverse the tree upwards until we hit the root dir
909 1          */
911 1         while (! (ptnPtr->tn_flags & TNP_ROOTDIR))
912 2         {
913 2             if (NULL == ptnPtr->tn_name)
914 3             {
915 3                 return(EP_RB_RECOVER_BAD_FREEMODE);
916 2             }
917 2             plen = strlen(ptnPtr->tn_name);
918 2             clen = strlen(fmptr);

920 2             /*
921 2              * Alloc buffer for concatenating the parent node name
922 2              * and the child node name, plus a leading '/' and a
923 2              * NULL-terminator
924 2              */
926 2             sPtr = fmptr; /* we'll need to free this buffer later */
927 2             fmptr = malloc(plen + clen + 2);
928 2             if (NULL == fmptr)
929 3             {
930 3                 rec_apl_log_cam(SUB_CSN_NOMEX, NULL);
931 3                 free(sPtr);
932 3                 return(EP_RB_RECOVER_NOMEX);
933 2             }

935 2             fmptr[0] = '/';
936 2             memcpy(fmptr+1, ptnPtr->tn_name, plen);

```

```

937 2             memcpy(fmptr+(1+plen), sPtr, clen);
938 2             fmptr[1+plen+clen] = 0;
940 2             free(sPtr);
942 2             ptnPtr = ptnPtr->tn_parent;
943 1         }
945 1         *fname = fmptr;
947 1         return(E_SUCCESS);
948 1         /* Concave\jllName */

```

Page 43 of 248	FindRealNode	Fri Jan 04 16:35:25 2008	Page 44 of 248	FindRealNode	Fri Jan 04 16:35:25 2008
<pre>950 /* 951 * Function: FindRealNode() 952 * 953 * Function Description: 954 * Given the ptr to a tree_node which is for a directory that 955 * does not have a valid corresponding catalog entry, this function 956 * traverses down the directory until it finds the subdir that 957 * has a valid catalog entry or until it reaches the end and fails. If 958 * it succeeds, the full pathname of the valid directory is 959 * returned. 960 * The caller should use get_strdup to copy the pathname. 961 * 962 * Parameters: 963 * trPtr - (T) ptr to the tree_node 964 * fname - (O) ptr to a pathname string ptr 965 * 966 * Return Codes: 967 * * SUCCESS 968 * * EP_RB_RECOVER_BAD_TREENODE - invalid treenode. 969 */ 970 971 static treeNode *trPtr; 972 973 FindRealNode (tree_node *trPtr, 974 char **fname) 975 { 976 tree_node *parentPtr; 977 tree_node *tmp; 978 int errnum; 979 treeNode elem; treeNode_t tree_elem; 980 treeNode_t catElem; 981 982 if (NULL == trPtr) 983 { 984 return (EP_RB_RECOVER_BAD_TREENODE); 985 } 986 987 /* 988 * The tree_node we are dealing with must be the first level 989 * down from the root dir '/'. If it's not, then return error. 990 * If the tree_node itself is not a directory, then it's also 991 * an error. 992 */ 993 994 parentPtr = trPtr->n.parent; 995 if (! (parentPtr->n.flags & TRF_ROOTDIR)) 996 { 997 return (EP_RB_RECOVER_BAD_TREENODE); 998 } 999 1000 if (! (trPtr->n.flags & TRF_ISDIR)) 1001 { 1002 return (EP_RB_RECOVER_BAD_TREENODE); 1003 } 1004 1005 tmp = trPtr; 1006 1007 /* 1008 * Traverse down the dir until a node is found that has a valid 1009 * catalog element. 1010 */ 1011 1012 for (;;) 1013 { 1014 /* 1015 * Looking for the top dir that has a valid catalog 1016 * entry. 1017 */ 1018 1019 count = mcat_array_children (tmp->rc_mcp, tmp, &dir_elem); 1020 mcat_array_risc (tmp->rc_mcp, dir_elem); 1021 1022 if (NULL == tmp) 1023 { 1024 return (EP_RB_RECOVER_BAD_TREENODE); 1025 } 1026 1027 if (! (tmp->n.flags & TRF_ISDIR)) 1028 { 1029 return (EP_RB_RECOVER_BAD_TREENODE); 1030 } 1031 1032 errnum = timo2reca (tmp, 1033 tmp->n.index, 1034 tmp->n.mcpname, 1035 &tree_elem, 1036 &catElem, 1037 fname); 1038 1039 if ((errnum == 0) && (fname != NULL)) 1040 { 1041 return (E_SUCCESS); 1042 } 1043 1044 if (count != 1) 1045 { 1046 return (EP_RB_RECOVER_BAD_TREENODE); 1047 } 1048 1049 /* FindRealNode */ 1050 } 1051 1052 }</pre>	<pre>1011 1 1012 1013 for (;;) 1014 { 1015 /* 1016 * Looking for the top dir that has a valid catalog 1017 * entry. 1018 */ 1019 1020 count = mcat_array_children (tmp->rc_mcp, tmp, &dir_elem); 1021 mcat_array_risc (tmp->rc_mcp, dir_elem); 1022 1023 if (NULL == tmp) 1024 { 1025 return (EP_RB_RECOVER_BAD_TREENODE); 1026 } 1027 1028 if (! (tmp->n.flags & TRF_ISDIR)) 1029 { 1030 return (EP_RB_RECOVER_BAD_TREENODE); 1031 } 1032 1033 errnum = timo2reca (tmp, 1034 tmp->n.index, 1035 tmp->n.mcpname, 1036 &tree_elem, 1037 &catElem, 1038 fname); 1039 1040 if ((errnum == 0) && (fname != NULL)) 1041 { 1042 return (E_SUCCESS); 1043 } 1044 1045 if (count != 1) 1046 { 1047 return (EP_RB_RECOVER_BAD_TREENODE); 1048 } 1049 1050 /* FindRealNode */ 1051 } 1052 1053 }</pre>				
Page 43 of 248	FindRealNode	Fri Jan 04 16:35:25 2008	Page 44 of 248	FindRealNode	Fri Jan 04 16:35:25 2008

```

1093     int
1094     fill_client_dirtop2(struct rbc_config *rconfig,
1095                        boolean_t by_inplace,
1096                        char *directory,
1097                        char *workitem_name,
1098                        char *dest_hostname,
1099                        char **client_dirtop)
1100 {
1101     char buf[4096];
1102     RBC_WORKGROUP *pwg;
1103     RBC_WORKITEM *pwi = NULL;
1104     boolean_t cross_recovery;
1105
1106     if (NULL == workitem_name) ||
1107         (NULL == client_dirtop)
1108     {
1109         return -1;
1110     }
1111     for (pwg = rconfig->pygroup_list; NULL != pwg; NULL == pwg->next)
1112     {
1113         for (pwi = pwg->pw_list; NULL != pwi; pwi = pwi->next)
1114         {
1115             if (0 == strcmp(pwi->name, workitem_name))
1116             {
1117                 break;
1118             }
1119         }
1120         if (NULL == pwi)
1121             return -1; /* didn't find workitem */
1122     }
1123
1124     /*
1125      * If the dirtop already specifies a network client
1126      * target lets return an error.
1127      */
1128     if (*client_dirtop != NULL
1129         && NULL != strchr(*client_dirtop, ':'))
1130     {
1131         return -1;
1132     }
1133
1134     /*
1135      * To test if this is a cross recovery compare the work items
1136      * configuration record field sysname (hostname) with the incoming
1137      * dest_hostname
1138      */
1139     cross_recovery = (NULL != pwi->sysname) && (
1140         (0 != strcmp(pwi->sysname, dest_hostname)) ||
1141         (char *temp_char="";
1142          if(NULL != pwi && NULL != pwi->nw_cint_target)
1143          {
1144              if(cross_recovery)
1145              {
1146                  temp_char = dest_hostname;
1147              }
1148          }
1149          else
1150          {
1151              temp_char = pwi->nw_cint_target;
1152          }
1153          if(0 != strcmp(temp_char, dest_hostname))
1154          {
1155              return -1;
1156          }
1157      ));
1158
1159     if (*client_dirtop != NULL)
1160     {
1161         free(*client_dirtop);
1162     }
1163
1164     *client_dirtop = esl_strdup(buf);
1165     if (NULL == *client_dirtop)
1166     {
1167         return -1;
1168     }
1169     return 0;
1170 }
1171
1172 /* fill_client_dirtop2 */

```

```

1116     }
1117     temp_char = pwi->nw_cint_target;
1118     sprintf(buf, "%s\\%s\\%s",
1119             temp_char,
1120             (
1121                 NULL != pwi && NULL != pwi->nw_cint_target ? ":" : "",
1122                 (NULL != directory) ? directory : "/"
1123             ));
1124
1125     if (*client_dirtop != NULL)
1126     {
1127         free(*client_dirtop);
1128     }
1129
1130     *client_dirtop = esl_strdup(buf);
1131     if (NULL == *client_dirtop)
1132     {
1133         return -1;
1134     }
1135     return 0;
1136 }
1137
1138 /* fill_client_dirtop2 */

```

```

1142 bool item_type GetListItemTypeFromConfig(struct RbcConfig *config,
1143                                         char *workitem_name,
1144                                         char *w1_type)
1145 {
1146     bool item_w1_found = FALSE;
1147     RBC_WORKGROUP *wgrp;
1148     RBC_WORKITEM *wip;
1149     if (NULL == workitem_name)
1150     {
1151         return FALSE;
1152     }
1153     for (wgrp = config->pgrouplist;
1154          wgrp != NULL; && (FALSE == w1_found) ;
1155          wgrp = wgrp->next)
1156     {
1157         RBC_WORKITEM *wip;
1158         for (wip = wgrp->wplist; wip != NULL; wip = wip->next)
1159         {
1160             if (NULL != wip->name) &&
1161                 (strcmp(wip->name, workitem_name) == 0))
1162             {
1163                 *w1_type = wip->w1_type;
1164                 w1_found = TRUE;
1165                 break;
1166             }
1167         }
1168     }
1169     return w1_found;
1170 }
1171
1172

```

[illegible]

```

1216 1      )
1216 1      return ret;
1217      }

/* make sure used */

1240
1241
1242
1243
1244
1245
1246
1247
1248
1249
1250
1251
1252
1253
1254
1255
1256
1257
1258
1259
1260
1261
1262
1263
1264
1265
1266
1267 1      {
1268 1          volumeId_t    internalVol;
1269 1          eerrno_t      ret = E_SUCCESS;
1270 1          ebv1_volumeId_t vol_entry;
1271 1
1272 1          if (NULL == volId || NULL == rcptr)
1273 1              return EP_RB_RECOVER_BAD_ARGS;
1274 1
1275 1          /* get volume id in internal format */
1276 1          removeFromAsciiToVolumeId(&internalVol, volId);
1277 1          if (NULL == (vol_entry =
1278 1              ebv1_is_volId_in_list(
1279 1                  &internalVol, rcptr->ebv1list)))
1280 1              {
1281 1                  ret = EP_RB_RECOVER_NO_VOLUME_DATA;
1282 1                  rcptr->internal_error_sub1
1283 1                      ret, "Error removing volume list entry" );
1284 1              }
1285 1          else
1286 1              {
1287 1                  /* decrement use count till zero */
1288 1                  while (DOES_VOLID_HAVR_MKD_FILES( vol_entry ))
1289 1                      DECREMENT_VOLID_FILE_COUNT( vol_entry );
1290 1              }
1291 1          return ret;
1292 1      }
1293

```

```

1295  */
1296  **
1297  **      RSTLL_FreeVolIdList
1298  **
1299  **      Frees a list of abv1_volIdList_cy's
1300  **
1301  **
1302  **      INPUTS:
1303  **      vol_list_ptr - address of the starting abv1_volIdList_cy entry
1304  **
1305  **      OUTPUTS:
1306  **      none
1307  **
1308  **      RETURN VALUE:
1309  **      none
1310  **
1311  **      IMPLICIT INPUTS:
1312  **
1313  **
1314  */
1315  void RSTLL_FreeVolIdList( IN abv1_volIdList_cy *vol_list_ptr )
1316  {
1317      abv1_volIdList_destructor( vol_list_ptr, RSTLL_DESTROY_ALL );
1318      return;
1319  }
1320

```

```

2  /*****
3  **
4  ** File Name:  RSUnmarkum.c
5  **
6  ** Copyright (c) 1998, 1999 by EMC Corporation.
7  **
8  **
9  ** Purpose:
10  **      This module contains the Restore Service Library functions to
11  **      unmark objects for restored.
12  **
13  ** -----
14  ** Table of Contents:
15  **
16  **      Service Library Functions:
17  **      RS256_UnmarkObject
18  **      RS256_UnmarkObject
19  **
20  ** Internal Functions:
21  **      int allowed_to_mark(
22  **          struct recover_context *rcx, tree_node *tmp,
23  **          char *name, if_known, rlocat_elem_t *clmp)
24  **
25  **      mkr(struct mocr_applyfunc_args *cxp)
26  **      unmk(struct mocr_applyfunc_args *cxp)
27  **      mark_tree(struct mocr_applyfunc_args *cxp)
28  **      struct recover_context
29  **      char *name, if_known)
30  **
31  **      unmark_tree(
32  **          struct recover_context *rcx, tree_node *tmp)
33  **
34  **
35  ** Compile-Time Options:
36  **      This section must list any compile time definitions
37  **      which will affect this header.
38  **
39  **
40  **
41  ** NOTES: Part of this module is adapted from:
42  **      server/libts/restore/grandfathered/cmd/markumark.c
43  **      It contains mainly support routines needed by the mark and unmark
44  **      API functions.
45  **
46  **
47  **
48  **
49  **
50  **
51  **
52  **
53  **
54  **
55  **
56  **
57  **
58  **
59  **
60  **
61  **
62  **
63  **
64  **
65  **
66  **
67  **
68  **
69  **
70  **
71  **
72  **
73  **
74  **
75  **
76  **
77  **
78  **
79  **
80  **
81  **
82  **
83  **
84  **
85  **
86  **
87  **
88  **
89  **
90  **
91  **
92  **
93  **
94  **
95  **
96  **
97  **
98  **
99  **
100  **
101  **
102  **
103  **
104  **
105  **
106  **
107  **
108  **
109  **
110  **
111  **
112  **
113  **
114  **
115  **
116  **
117  **
118  **
119  **
120  **
121  **
122  **
123  **
124  **
125  **
126  **
127  **
128  **
129  **
130  **
131  **
132  **
133  **
134  **
135  **
136  **
137  **
138  **
139  **
140  **
141  **
142  **
143  **
144  **
145  **
146  **
147  **
148  **
149  **
150  **
151  **
152  **
153  **
154  **
155  **
156  **
157  **
158  **
159  **
160  **
161  **
162  **
163  **
164  **
165  **
166  **
167  **
168  **
169  **
170  **
171  **
172  **
173  **
174  **
175  **
176  **
177  **
178  **
179  **
180  **
181  **
182  **
183  **
184  **
185  **
186  **
187  **
188  **
189  **
190  **
191  **
192  **
193  **
194  **
195  **
196  **
197  **
198  **
199  **
200  **
201  **
202  **
203  **
204  **
205  **
206  **
207  **
208  **
209  **
210  **
211  **
212  **
213  **
214  **
215  **
216  **
217  **
218  **
219  **
220  **
221  **
222  **
223  **
224  **
225  **
226  **
227  **
228  **
229  **
230  **
231  **
232  **
233  **
234  **
235  **
236  **
237  **
238  **
239  **
240  **
241  **
242  **
243  **
244  **
245  **
246  **
247  **
248  **
249  **
250  **
251  **
252  **
253  **
254  **
255  **
256  **
257  **
258  **
259  **
260  **
261  **
262  **
263  **
264  **
265  **
266  **
267  **
268  **
269  **
270  **
271  **
272  **
273  **
274  **
275  **
276  **
277  **
278  **
279  **
280  **
281  **
282  **
283  **
284  **
285  **
286  **
287  **
288  **
289  **
290  **
291  **
292  **
293  **
294  **
295  **
296  **
297  **
298  **
299  **
300  **
301  **
302  **
303  **
304  **
305  **
306  **
307  **
308  **
309  **
310  **
311  **
312  **
313  **
314  **
315  **
316  **
317  **
318  **
319  **
320  **
321  **
322  **
323  **
324  **
325  **
326  **
327  **
328  **
329  **
330  **
331  **
332  **
333  **
334  **
335  **
336  **
337  **
338  **
339  **
340  **
341  **
342  **
343  **
344  **
345  **
346  **
347  **
348  **
349  **
350  **
351  **
352  **
353  **
354  **
355  **
356  **
357  **
358  **
359  **
360  **
361  **
362  **
363  **
364  **
365  **
366  **
367  **
368  **
369  **
370  **
371  **
372  **
373  **
374  **
375  **
376  **
377  **
378  **
379  **
380  **
381  **
382  **
383  **
384  **
385  **
386  **
387  **
388  **
389  **
390  **
391  **
392  **
393  **
394  **
395  **
396  **
397  **
398  **
399  **
400  **
401  **
402  **
403  **
404  **
405  **
406  **
407  **
408  **
409  **
410  **
411  **
412  **
413  **
414  **
415  **
416  **
417  **
418  **
419  **
420  **
421  **
422  **
423  **
424  **
425  **
426  **
427  **
428  **
429  **
430  **
431  **
432  **
433  **
434  **
435  **
436  **
437  **
438  **
439  **
440  **
441  **
442  **
443  **
444  **
445  **
446  **
447  **
448  **
449  **
450  **
451  **
452  **
453  **
454  **
455  **
456  **
457  **
458  **
459  **
460  **
461  **
462  **
463  **
464  **
465  **
466  **
467  **
468  **
469  **
470  **
471  **
472  **
473  **
474  **
475  **
476  **
477  **
478  **
479  **
480  **
481  **
482  **
483  **
484  **
485  **
486  **
487  **
488  **
489  **
490  **
491  **
492  **
493  **
494  **
495  **
496  **
497  **
498  **
499  **
500  **
501  **
502  **
503  **
504  **
505  **
506  **
507  **
508  **
509  **
510  **
511  **
512  **
513  **
514  **
515  **
516  **
517  **
518  **
519  **
520  **
521  **
522  **
523  **
524  **
525  **
526  **
527  **
528  **
529  **
530  **
531  **
532  **
533  **
534  **
535  **
536  **
537  **
538  **
539  **
540  **
541  **
542  **
543  **
544  **
545  **
546  **
547  **
548  **
549  **
550  **
551  **
552  **
553  **
554  **
555  **
556  **
557  **
558  **
559  **
560  **
561  **
562  **
563  **
564  **
565  **
566  **
567  **
568  **
569  **
570  **
571  **
572  **
573  **
574  **
575  **
576  **
577  **
578  **
579  **
580  **
581  **
582  **
583  **
584  **
585  **
586  **
587  **
588  **
589  **
590  **
591  **
592  **
593  **
594  **
595  **
596  **
597  **
598  **
599  **
600  **
601  **
602  **
603  **
604  **
605  **
606  **
607  **
608  **
609  **
610  **
611  **
612  **
613  **
614  **
615  **
616  **
617  **
618  **
619  **
620  **
621  **
622  **
623  **
624  **
625  **
626  **
627  **
628  **
629  **
630  **
631  **
632  **
633  **
634  **
635  **
636  **
637  **
638  **
639  **
640  **
641  **
642  **
643  **
644  **
645  **
646  **
647  **
648  **
649  **
650  **
651  **
652  **
653  **
654  **
655  **
656  **
657  **
658  **
659  **
660  **
661  **
662  **
663  **
664  **
665  **
666  **
667  **
668  **
669  **
670  **
671  **
672  **
673  **
674  **
675  **
676  **
677  **
678  **
679  **
680  **
681  **
682  **
683  **
684  **
685  **
686  **
687  **
688  **
689  **
690  **
691  **
692  **
693  **
694  **
695  **
696  **
697  **
698  **
699  **
700  **
701  **
702  **
703  **
704  **
705  **
706  **
707  **
708  **
709  **
710  **
711  **
712  **
713  **
714  **
715  **
716  **
717  **
718  **
719  **
720  **
721  **
722  **
723  **
724  **
725  **
726  **
727  **
728  **
729  **
730  **
731  **
732  **
733  **
734  **
735  **
736  **
737  **
738  **
739  **
740  **
741  **
742  **
743  **
744  **
745  **
746  **
747  **
748  **
749  **
750  **
751  **
752  **
753  **
754  **
755  **
756  **
757  **
758  **
759  **
760  **
761  **
762  **
763  **
764  **
765  **
766  **
767  **
768  **
769  **
770  **
771  **
772  **
773  **
774  **
775  **
776  **
777  **
778  **
779  **
780  **
781  **
782  **
783  **
784  **
785  **
786  **
787  **
788  **
789  **
790  **
791  **
792  **
793  **
794  **
795  **
796  **
797  **
798  **
799  **
800  **
801  **
802  **
803  **
804  **
805  **
806  **
807  **
808  **
809  **
810  **
811  **
812  **
813  **
814  **
815  **
816  **
817  **
818  **
819  **
820  **
821  **
822  **
823  **
824  **
825  **
826  **
827  **
828  **
829  **
830  **
831  **
832  **
833  **
834  **
835  **
836  **
837  **
838  **
839  **
840  **
841  **
842  **
843  **
844  **
845  **
846  **
847  **
848  **
849  **
850  **
851  **
852  **
853  **
854  **
855  **
856  **
857  **
858  **
859  **
860  **
861  **
862  **
863  **
864  **
865  **
866  **
867  **
868  **
869  **
870  **
871  **
872  **
873  **
874  **
875  **
876  **
877  **
878  **
879  **
880  **
881  **
882  **
883  **
884  **
885  **
886  **
887  **
888  **
889  **
890  **
891  **
892  **
893  **
894  **
895  **
896  **
897  **
898  **
899  **
900  **
901  **
902  **
903  **
904  **
905  **
906  **
907  **
908  **
909  **
910  **
911  **
912  **
913  **
914  **
915  **
916  **
917  **
918  **
919  **
920  **
921  **
922  **
923  **
924  **
925  **
926  **
927  **
928  **
929  **
930  **
931  **
932  **
933  **
934  **
935  **
936  **
937  **
938  **
939  **
940  **
941  **
942  **
943  **
944  **
945  **
946  **
947  **
948  **
949  **
950  **
951  **
952  **
953  **
954  **
955  **
956  **
957  **
958  **
959  **
960  **
961  **
962  **
963  **
964  **
965  **
966  **
967  **
968  **
969  **
970  **
971  **
972  **
973  **
974  **
975  **
976  **
977  **
978  **
979  **
980  **
981  **
982  **
983  **
984  **
985  **
986  **
987  **
988  **
989  **
990  **
991  **
992  **
993  **
994  **
995  **
996  **
997  **
998  **
999  **
1000  **

```

```

60  #define _POSIX_SOURCE 1
61
62  /*
63  **
64  ** System headers.
65  **
66  **
67  **
68  **
69  ** Epoch headers.
70  **
71  ** #include <eb/eb_port.h>
72  ** #include <eb/rb_log.h>
73  **
74  **
75  ** Local headers
76  **
77  ** #include <RSUnmark.h>
78  ** #include <RSUnmarkmain.h>
79  **
80  **
81  ** #defines, structures, typedefs local to this source file
82  **
83  **
84  **
85  **
86  ** struct mark_context
87  ** {
88  **     boolean_t no_badfiles; /* don't mark bad files */
89  **     boolean_t no_descend; /* don't descend dirs, just mark them */
90  **     ulong_t n_this;
91  **     ulong_t n_before;
92  ** };
93  **
94  **
95  ** static int
96  ** mark_tree(struct recover_context *rcx,
97  **     tree_node *tmp,
98  **     char *name, if_known);
99  **
100  **
101  ** static int
102  ** unmark_tree(struct recover_context *rcx,
103  **     tree_node *tmp);
104  **
105  **
106  **
107  **
108  **
109  **
110  **
111  **
112  **
113  **
114  **
115  **
116  **
117  **
118  **
119  **
120  **
121  **
122  **
123  **
124  **
125  **
126  **
127  **
128  **
129  **
130  **
131  **
132  **
133  **
134  **
135  **
136  **
137  **
138  **
139  **
140  **
141  **
142  **
143  **
144  **
145  **
146  **
147  **
148  **
149  **
150  **
151  **
152  **
153  **
154  **
155  **
156  **
157  **
158  **
159  **
160  **
161  **
162  **
163  **
164  **
165  **
166  **
167  **
168  **
169  **
170  **
171  **
172  **
173  **
174  **
175  **
176  **
177  **
178  **
179  **
180  **
181  **
182  **
183  **
184  **
185  **
186  **
187  **
188  **
189  **
190  **
191  **
192  **
193  **
194  **
195  **
196  **
197  **
198  **
199  **
200  **
201  **
202  **
203  **
204  **
205  **
206  **
207  **
208  **
209  **
210  **
211  **
212  **
213  **
214  **
215  **
216  **
217  **
218  **
219  **
220  **
221  **
222  **
223  **
224  **
225  **
226  **
227  **
228  **
229  **
230  **
231  **
232  **
233  **
234  **
235  **
236  **
237  **
238  **
239  **
240  **
241  **
242  **
243  **
244  **
245  **
246  **
247  **
248  **
249  **
250  **
251  **
252  **
253  **
254  **
255  **
256  **
257  **
258  **
259  **
260  **
261  **
262  **
263  **
264  **
265  **
266  **
267  **
268  **
269  **
270  **
271  **
272  **
273  **
274  **
275  **
276  **
277  **
278  **
279  **
280  **
281  **
282  **
283  **
284  **
285  **
286  **
287  **
288  **
289  **
290  **
291  **
292  **
293  **
294  **
295  **
296  **
297  **
298  **
299  **
300  **
301  **
302  **
303  **
304  **
305  **
306  **
307  **
308  **
309  **
310  **
311  **
312  **
313  **
314  **
315  **
316  **
317  **
318  **
319  **
320  **
321  **
322  **
323  **
324  **
325  **
326  **
327  **
328  **
329  **
330  **
331  **
332  **
333  **
334  **
335  **
336  **
337  **
338  **
339  **
340  **
341  **
342  **
343  **
344  **
345  **
346  **
347  **
348  **
349  **
350  **
351  **
352  **
353  **
354  **
355  **
356  **
357  **
358  **
359  **
360  **
361  **
362  **
363  **
364  **
365  **
366  **
367  **
368  **
369  **
370  **
371  **
372  **
373  **
374  **
375  **
376  **
377  **
378  **
379  **
380  **
381  **
382  **
383  **
384  **
385  **
386  **
387  **
388  **
389  **
390  **
391  **
392  **
393  **
394  **
395  **
396  **
397  **
398  **
399  **
400  **
401  **
402  **
403  **
404  **
405  **
406  **
407  **
408  **
409  **
410  **
411  **
412  **
413  **
414  **
415  **
416  **
417  **
418  **
419  **
420  **
421  **
422  **
423  **
424  **
425  **
426  **
427  **
428  **
429  **
430  **
431  **
432  **
433  **
434  **
435  **
436  **
437  **
438  **
439  **
440  **
441  **
442  **
443  **
444  **
445  **
446  **
447  **
448  **
449  **
450  **
451  **
452  **
453  **
454  **
455  **
456  **
457  **
458  **
459  **
460  **
461  **
462  **
463  **
464  **
465  **
466  **
467  **
468  **
469  **
470  **
471  **
472  **
473  **
474  **
475  **
476  **
477  **
478  **
479  **
480  **
481  **
482  **
483  **
484  **
485  **
486  **
487  **
488  **
489  **
490  **
491  **
492  **
493  **
494  **
495  **
496  **
497  **
498  **
499  **
500  **
501  **
502  **
503  **
504  **
505  **
506  **
507  **
508  **
509  **
510  **
511  **
512  **
513  **
514  **
515  **
516  **
517  **
518  **
519  **
520  **
521  **
522  **
523  **
524  **
525  **
526  **
527  **
528  **
529  **
530  **
531  **
532  **
533  **
534  **
535  **
536  **
537  **
538  **
539  **
540  **
541  **
542  **
543  **
544  **
545  **
546  **
547  **
548  **
549  **
550  **
551  **
552  **
553  **
554  **
555  **
556  **
557  **
558  **
559  **
560  **
561  **
562  **
563  **
564  **
565  **
566  **
567  **
568  **
569  **
570  **
571  **
572  **
573  **
574  **
575  **
576  **
577  **
578  **
579  **
580  **
581  **
582  **
583  **
584  **
585  **
586  **
587  **
588  **
589  **
590  **
591  **
592  **
593  **
594  **
595  **
596  **
597  **
598  **
599  **
600  **
601  **
602  **
603  **
604  **
605  **
606  **
607  **
608  **
609  **
610  **
611  **
612  **
613  **
614  **
615  **
616  **
617  **
618  **
619  **
620  **
621  **
622  **
623  **
624  **
625  **
626  **
627  **
628  **
629  **
630  **
631  **
632  **
633  **
634  **
635  **
636  **
637  **
638  **
639  **
640  **
641  **
642  **
643  **
644  **
645  **
646  **
647  **
648  **
649  **
650  **
651  **
652  **
653  **
654  **
655  **
656  **
657  **
658  **
659  **
660  **
661  **
662  **
663  **
664  **
665  **
666  **
667  **
668  **
669  **
670  **
671  **
672  **
673  **
674  **
675  **
676  **
677  **
678  **
679  **
680  **
681  **
682  **
683  **
684  **
685  **
686  **
687  **
688  **
689  **
690  **
691  **
692  **
693  **
694  **
695  **
696  **
697  **
698  **
699  **
700  **
701  **
702  **
703  **
704  **
705  **
706  **
707  **
708  **
709  **
710  **
711  **
712  **
713  **
714  **
715  **
716  **
717  **
718  **
719  **
720  **
721  **
722  **
723  **
724  **
725  **
726  **
727  **
728  **
729  **
730  **
731  **
732  **
733  **
734  **
735  **
736  **
737  **
738  **
739  **
740  **
741  **
742  **
743  **
744  **
745  **
746  **
747  **
748  **
749  **
750  **
751  **
752  **
753  **
754  **
755  **
756  **
757  **
758  **
759  **
760  **
761  **
762  **
763  **
764  **
765  **
766  **
767  **
768  **
769  **
770  **
771  **
772  **
773  **
774  **
775  **
776  **
777  **
778  **
779  **
780  **
781  **
782  **
783  **
784  **
785  **
786  **
787  **
788  **
789  **
790  **
791  **
792  **
793  **
794  **
795  **
796  **
797  **
798  **
799  **
800  **
801  **
802  **
803  **
804  **
805  **
806  **
807  **
808  **
809  **
810  **
811  **
812  **
813  **
814  **
815  **
816  **
817  **
818  **
819  **
820  **
821  **
822  **
823  **
824  **
825  **
826  **
827  **
828  **
829  **
830  **
831  **
832  **
833  **
834  **
835  **
836  **
837  **
838  **
839  **
840  **
841  **
842  **
843  **
844  **
845  **
846  **
847  **
848  **
849  **
850  **
851  **
852  **
853  **
854  **
855  **
856  **
857  **
858  **
859  **
860  **
861  **
862  **
863  **
864  **
865  **
866  **
867  **
868  **
869  **
870  **
871  **
872  **
873  **
874  **
875  **
876  **
877  **
878  **
879  **
880  **
881  **
882  **
883  **
884  **
885  **
886  **
887  **
888  **
889  **
890  **
891  **
892  **
893  **
894  **
895  **
896  **
897  **
898  **
899  **
900  **

```



```

125 static int check_parent_perms(struct recover_context *rcx,
126 char *name, if_known, char *errorbuff);

```

```

128 /*
129 .....
130 * allowed to mark:
131 *
132 * Does the recovering user have permission to a file
133 * for recovery.
134 *
135 * returns
136 * 1 if allowed to mark, 0 if permission is denied.
137 *
138 * Parameters:
139 * rcx (I)
140 * tnp (I) -- tree node pointer
141 * name, if_known (I) -- name of file
142 * clmp (I) -- catalog elem
143 .....
144 */

```

```

145 int
146 allowed_to_mark(struct recover_context *rcx,
147 tree_node
148 char
149 *rbcat_elem_t
150 *clmp)
151 {
152     int ok = 1;

```

```

153     /*
154     * Modification for bug no OSGSW 23406
155     * perm_reason initialized to empty string
156     */
157     char perm_reason[256] = "";
158     /*
159     * Modifications for bug no OSGSW23406 end
160     */

```

```

161     if (S_ISREG(clmp->ce_mode))
162     {
163         /*
164         * For regular files read permission required.
165         */
166         ok = rcx_permchk(rcx, PERMCHK_R, tnp, &clmp);
167     }
168     else if (S_ISDIR(clmp->ce_mode))
169     {
170         /*
171         * For directories execute and read permission required.
172         */
173         ok = rcx_permchk(rcx, PERMCHK_R|PERMCHK_X, tnp, &clmp);
174     }
175     else if (S_ISBLK(clmp->ce_mode) || S_ISCHR(clmp->ce_mode))
176     {
177         /*
178         * For char and block devices,
179         * only super-user can extract these.
180         */
181         ok = (rcx->c_effective_uid == 0);
182         strcpy(perm_reason, "to recover you must be super-user");
183     }
184     else
185     {
186         /*
187         * For unexpected file types no permission checks and we log
188         * this one.
189         */

```

```

192 2 */
194 2 char pathbuf[EB_MAXPATHLEN];
195 2 char *pbp = pathbuf;
198 2
199 2    rn.getpath(tmp, &pbp);
200 2
201 2    ok = 1;
202 2    rbe_log_stats(0,
203 2        "Unexpected file type,
204 2        no permission checks to mark file %s", pbp);
205 2
206 2    )
207 2
208 2    if (!ok)
209 2    {
210 2        /*
211 2        * Modifications for bug no OSGsw23406
212 2        */
213 2        if (NULL != name_if_known)
214 2        {
215 2            rbe_log_stats(0,
216 2                "Permission denied for file '%s' %s.",
217 2                name_if_known, perm_reason);
218 2
219 2            )
220 2            {
221 2                char pathbuf[EB_MAXPATHLEN];
222 2                char *pbp = pathbuf;
223 2
224 2                rn.getpath(tmp, &pbp);
225 2                rbe_log_stats(0,
226 2                    "Permission denied for file '%s' %s.",
227 2                    pbp, perm_reason);
228 2
229 2                /*
230 2                * Modifications for bug no OSGsw23406 end
231 2                */
232 2            }
233 2
234 2            /*
235 2            * end of if not ok */
236 2
237 2            return OK;
238 2
239 2            /*
240 2            * end of allowed_to_mark() */
241 2

```

```

237 2
238 2
239 2    * mtx:
240 2
241 2    *
242 2    *
243 2    *
244 2    *
245 2    *
246 2    *
247 2    *
248 2    *
249 2    *
250 2    *
251 2    *
252 2    *
253 2    *
254 2    *
255 2    *
256 2    *
257 2    *
258 2    *
259 2    *
260 2    *
261 2    *
262 2    *
263 2    *
264 2    *
265 2    *
266 2    *
267 2    *
268 2    *
269 2    *
270 2    *
271 2    *
272 2    *
273 2    *
274 2    *
275 2    *
276 2    *
277 2    *
278 2    *
279 2    *
280 2    *
281 2    *
282 2    *
283 2    *
284 2    *
285 2    *
286 2    *
287 2    *
288 2    *
289 2    *
290 2    *
291 2    *
292 2    *
293 2    *
294 2    *
295 2    *
296 2    *
297 2    *
298 2    *
299 2    *
300 2    *
301 2    *
302 2    *
303 2    *
304 2    *
305 2    *
306 2    *
307 2    *
308 2    *
309 2    *
310 2    *
311 2    *
312 2    *
313 2    *
314 2    *
315 2    *
316 2    *
317 2    *
318 2    *
319 2    *
320 2    *
321 2    *
322 2    *
323 2    *
324 2    *
325 2    *
326 2    *
327 2    *
328 2    *
329 2    *
330 2    *
331 2    *
332 2    *
333 2    *
334 2    *
335 2    *
336 2    *
337 2    *
338 2    *
339 2    *
340 2    *
341 2    *
342 2    *
343 2    *
344 2    *
345 2    *
346 2    *
347 2    *
348 2    *
349 2    *
350 2    *
351 2    *
352 2    *
353 2    *
354 2    *
355 2    *
356 2    *
357 2    *
358 2    *
359 2    *
360 2    *
361 2    *
362 2    *
363 2    *
364 2    *
365 2    *
366 2    *
367 2    *
368 2    *
369 2    *
370 2    *
371 2    *
372 2    *
373 2    *
374 2    *
375 2    *
376 2    *
377 2    *
378 2    *
379 2    *
380 2    *
381 2    *
382 2    *
383 2    *
384 2    *
385 2    *
386 2    *
387 2    *
388 2    *
389 2    *
390 2    *
391 2    *
392 2    *
393 2    *
394 2    *
395 2    *
396 2    *
397 2    *
398 2    *
399 2    *
400 2    *
401 2    *
402 2    *
403 2    *
404 2    *
405 2    *
406 2    *
407 2    *
408 2    *
409 2    *
410 2    *
411 2    *
412 2    *
413 2    *
414 2    *
415 2    *
416 2    *
417 2    *
418 2    *
419 2    *
420 2    *
421 2    *
422 2    *
423 2    *
424 2    *
425 2    *
426 2    *
427 2    *
428 2    *
429 2    *
430 2    *
431 2    *
432 2    *
433 2    *
434 2    *
435 2    *
436 2    *
437 2    *
438 2    *
439 2    *
440 2    *
441 2    *
442 2    *
443 2    *
444 2    *
445 2    *
446 2    *
447 2    *
448 2    *
449 2    *
450 2    *
451 2    *
452 2    *
453 2    *
454 2    *
455 2    *
456 2    *
457 2    *
458 2    *
459 2    *
460 2    *
461 2    *
462 2    *
463 2    *
464 2    *
465 2    *
466 2    *
467 2    *
468 2    *
469 2    *
470 2    *
471 2    *
472 2    *
473 2    *
474 2    *
475 2    *
476 2    *
477 2    *
478 2    *
479 2    *
480 2    *
481 2    *
482 2    *
483 2    *
484 2    *
485 2    *
486 2    *
487 2    *
488 2    *
489 2    *
490 2    *
491 2    *
492 2    *
493 2    *
494 2    *
495 2    *
496 2    *
497 2    *
498 2    *
499 2    *
500 2    *
501 2    *
502 2    *
503 2    *
504 2    *
505 2    *
506 2    *
507 2    *
508 2    *
509 2    *
510 2    *
511 2    *
512 2    *
513 2    *
514 2    *
515 2    *
516 2    *
517 2    *
518 2    *
519 2    *
520 2    *
521 2    *
522 2    *
523 2    *
524 2    *
525 2    *
526 2    *
527 2    *
528 2    *
529 2    *
530 2    *
531 2    *
532 2    *
533 2    *
534 2    *
535 2    *
536 2    *
537 2    *
538 2    *
539 2    *
540 2    *
541 2    *
542 2    *
543 2    *
544 2    *
545 2    *
546 2    *
547 2    *
548 2    *
549 2    *
550 2    *
551 2    *
552 2    *
553 2    *
554 2    *
555 2    *
556 2    *
557 2    *
558 2    *
559 2    *
560 2    *
561 2    *
562 2    *
563 2    *
564 2    *
565 2    *
566 2    *
567 2    *
568 2    *
569 2    *
570 2    *
571 2    *
572 2    *
573 2    *
574 2    *
575 2    *
576 2    *
577 2    *
578 2    *
579 2    *
580 2    *
581 2    *
582 2    *
583 2    *
584 2    *
585 2    *
586 2    *
587 2    *
588 2    *
589 2    *
590 2    *
591 2    *
592 2    *
593 2    *
594 2    *
595 2    *
596 2    *
597 2    *
598 2    *
599 2    *
600 2    *
601 2    *
602 2    *
603 2    *
604 2    *
605 2    *
606 2    *
607 2    *
608 2    *
609 2    *
610 2    *
611 2    *
612 2    *
613 2    *
614 2    *
615 2    *
616 2    *
617 2    *
618 2    *
619 2    *
620 2    *
621 2    *
622 2    *
623 2    *
624 2    *
625 2    *
626 2    *
627 2    *
628 2    *
629 2    *
630 2    *
631 2    *
632 2    *
633 2    *
634 2    *
635 2    *
636 2    *
637 2    *
638 2    *
639 2    *
640 2    *
641 2    *
642 2    *
643 2    *
644 2    *
645 2    *
646 2    *
647 2    *
648 2    *
649 2    *
650 2    *
651 2    *
652 2    *
653 2    *
654 2    *
655 2    *
656 2    *
657 2    *
658 2    *
659 2    *
660 2    *
661 2    *
662 2    *
663 2    *
664 2    *
665 2    *
666 2    *
667 2    *
668 2    *
669 2    *
670 2    *
671 2    *
672 2    *
673 2    *
674 2    *
675 2    *
676 2    *
677 2    *
678 2    *
679 2    *
680 2    *
681 2    *
682 2    *
683 2    *
684 2    *
685 2    *
686 2    *
687 2    *
688 2    *
689 2    *
690 2    *
691 2    *
692 2    *
693 2    *
694 2    *
695 2    *
696 2    *
697 2    *
698 2    *
699 2    *
700 2    *
701 2    *
702 2    *
703 2    *
704 2    *
705 2    *
706 2    *
707 2    *
708 2    *
709 2    *
710 2    *
711 2    *
712 2    *
713 2    *
714 2    *
715 2    *
716 2    *
717 2    *
718 2    *
719 2    *
720 2    *
721 2    *
722 2    *
723 2    *
724 2    *
725 2    *
726 2    *
727 2    *
728 2    *
729 2    *
730 2    *
731 2    *
732 2    *
733 2    *
734 2    *
735 2    *
736 2    *
737 2    *
738 2    *
739 2    *
740 2    *
741 2    *
742 2    *
743 2    *
744 2    *
745 2    *
746 2    *
747 2    *
748 2    *
749 2    *
750 2    *
751 2    *
752 2    *
753 2    *
754 2    *
755 2    *
756 2    *
757 2    *
758 2    *
759 2    *
760 2    *
761 2    *
762 2    *
763 2    *
764 2    *
765 2    *
766 2    *
767 2    *
768 2    *
769 2    *
770 2    *
771 2    *
772 2    *
773 2    *
774 2    *
775 2    *
776 2    *
777 2    *
778 2    *
779 2    *
780 2    *
781 2    *
782 2    *
783 2    *
784 2    *
785 2    *
786 2    *
787 2    *
788 2    *
789 2    *
790 2    *
791 2    *
792 2    *
793 2    *
794 2    *
795 2    *
796 2    *
797 2    *
798 2    *
799 2    *
800 2    *
801 2    *
802 2    *
803 2    *
804 2    *
805 2    *
806 2    *
807 2    *
808 2    *
809 2    *
810 2    *
811 2    *
812 2    *
813 2    *
814 2    *
815 2    *
816 2    *
817 2    *
818 2    *
819 2    *
820 2    *
821 2    *
822 2    *
823 2    *
824 2    *
825 2    *
826 2    *
827 2    *
828 2    *
829 2    *
830 2    *
831 2    *
832 2    *
833 2    *
834 2    *
835 2    *
836 2    *
837 2    *
838 2    *
839 2    *
840 2    *
841 2    *
842 2    *
843 2    *
844 2    *
845 2    *
846 2    *
847 2    *
848 2    *
849 2    *
850 2    *
851 2    *
852 2    *
853 2    *
854 2    *
855 2    *
856 2    *
857 2    *
858 2    *
859 2    *
860 2    *
861 2    *
862 2    *
863 2    *
864 2    *
865 2    *
866 2    *
867 2    *
868 2    *
869 2    *
870 2    *
871 2    *
872 2    *
873 2    *
874 2    *
875 2    *
876 2    *
877 2    *
878 2    *
879 2    *
880 2    *
881 2    *
882 2    *
883 2    *
884 2    *
885 2    *
886 2    *
887 2    *
888 2    *
889 2    *
890 2    *
891 2    *
892 2    *
893 2    *
894 2    *
895 2    *
896 2    *
897 2    *
898 2    *
899 2    *
900 2    *
901 2    *
902 2    *
903 2    *
904 2    *
905 2    *
906 2    *
907 2    *
908 2    *
909 2    *
910 2    *
911 2    *
912 2    *
913 2    *
914 2    *
915 2    *
916 2    *
917 2    *
918 2    *
919 2    *
920 2    *
921 2    *
922 2    *
923 2    *
924 2    *
925 2    *
926 2    *
927 2    *
928 2    *
929 2    *
930 2    *
931 2    *
932 2    *
933 2    *
934 2    *
935 2    *
936 2    *
937 2    *
938 2    *
939 2    *
940 2    *
941 2    *
942 2    *
943 2    *
944 2    *
945 2    *
946 2    *
947 2    *
948 2    *
949 2    *
950 2    *
951 2    *
952 2    *
953 2    *
954 2    *
955 2    *
956 2    *
957 2    *
958 2    *
959 2    *
960 2    *
961 2    *
962 2    *
963 2    *
964 2    *
965 2    *
966 2    *
967 2    *
968 2    *
969 2    *
970 2    *
971 2    *
972 2    *
973 2    *
974 2    *
975 2    *
976 2    *
977 2    *
978 2    *
979 2    *
980 2    *
981 2    *
982 2    *
983 2    *
984 2    *
985 2    *
986 2    *
987 2    *
988 2    *
989 2    *
990 2    *
991 2    *
992 2    *
993 2    *
994 2    *
995 2    *
996 2    *
997 2    *
998 2    *
999 2    *
1000 2    *

```

```

313 /*.....
314 *
315 * unix:
316 *
317 *
318 *.....
319
320 static int
321 unix(struct mcat_apply_func_args *ctcp)
322 {
323     struct recover_context *rcx =
324         LINTABLE_CAST(struct recover_context *, ctcp->arg);
325
326     return unmark_tree(rcx, ctcp->tmp);
327 }
328 /* end of unix() */

```

```

319 /*
320 * The below are static variables global to this module ap_l_markmark.
321 * They get initialized in RSTSL_MarkObjec to 0.
322 * They get incremented in mark_tree() when the conditions of mark
323 * are encountered
324 * They get assigned to the output parameters to RSTSL_MarkObjec
325 * after
326 * the mark_tree call.
327
328 static unsigned long PermissionsDeniedIflCount;
329 static unsigned long BDDAPIflCount;
330
331 /*.....
332 *
333 * mark_tree:
334 *
335 * Provides mark function that is based on the current mcat
336 * recover context. The mark operation has hidden parameters
337 * in the mark_context struct.
338 *
339 * Globals:
340 * PermissionsDeniedIflCount and BDDAPIflCount
341 * are statics global to this module.
342 *
343 * returns 0 for success always.
344
345 * Parameters:
346 * rcx -- recover context
347 * tmp (I) -- tree node pointer
348 * name, if known (I) -- name of input file.
349 *.....
350
351 static int
352 mark_tree(struct recover_context *rcx,
353            tree_node *tmp,
354            char *name, if_known)
355 {
356     struct mark_context *mcp =
357         LINTABLE_CAST(struct mark_context *, rcx->rc_ctx);
358
359     eperrno ep_status = E_SUCCESS;

```

```

319
320
321 /*
322 * Check to see if the mark operation has been cancelled.
323 * If so, return without marking.
324 *
325 * This is due to the recursiveness of the mcat_apply2children
326 * function to stop any further processing.
327 */
328
329 if (global_was_cancelled)
330     return (0);
331
332 /*
333 * Only do work if element not already marked.
334 */
335 if (! MARKED(rcx->rc_marks, tmp))
336 {

```

```

311. 2      libcat_olam.c: cfm;
313. 2      /*
314. 2      * get catalog record
315. 2      */
317. 2      mact_getcat(mtx->cmap, tmp, &cm);
319. 2
320. 2      /*
321. 2      * Skip file if it's bad and user wants to ignore bad files
322. 2      */
323. 2      if (tmp->badfiles && (CSTATUS_BADDATA & cm.ce_status))
324. 2      {
325. 2          char pathbuf[PB_MAXPATHLEN];
326. 2          char *p = pathbuf;
327. 2
328. 2          if (CSTATUS_BADDATA & cm.ce_status)
329. 2          {
330. 2              if (getpath(tmp, &pp) :
331. 2                  re_log_stats(
332. 2                      0, "the file %s has BADDATA and was not marked", &pp);
333. 2              BADDATAfilecount++;
334. 2              return 0;
335. 2          }
336. 2          if (CSTATUS_BADDATA & cm.ce_status)
337. 2          {
338. 2              char pathbuf[PB_MAXPATHLEN];
339. 2              char *p = pathbuf;
340. 2
341. 2              if (getpath(tmp, &pp) :
342. 2                  re_log_stats(
343. 2                      0, "the file %s has BADDATA and was marked", &pp);
344. 2              BADDATAfilecount++;
345. 2              return 0;
346. 2          }
347. 2          if (CSTATUS_BADDATA & cm.ce_status)
348. 2          {
349. 2              char pathbuf[PB_MAXPATHLEN];
350. 2              char *p = pathbuf;
351. 2
352. 2              if (getpath(tmp, &pp) :
353. 2                  re_log_stats(
354. 2                      0, "the file %s has BADDATA and was marked", &pp);
355. 2              BADDATAfilecount++;
356. 2              return 0;
357. 2          }
358. 2          if (CSTATUS_BADDATA & cm.ce_status)
359. 2          {
360. 2              char pathbuf[PB_MAXPATHLEN];
361. 2              char *p = pathbuf;
362. 2
363. 2              if (getpath(tmp, &pp) :
364. 2                  re_log_stats(
365. 2                      0, "the file %s has BADDATA and was marked", &pp);
366. 2              BADDATAfilecount++;
367. 2              return 0;
368. 2          }
369. 2          if (CSTATUS_BADDATA & cm.ce_status)
370. 2          {
371. 2              char pathbuf[PB_MAXPATHLEN];
372. 2              char *p = pathbuf;
373. 2
374. 2              if (getpath(tmp, &pp) :
375. 2                  re_log_stats(
376. 2                      0, "the file %s has BADDATA and was marked", &pp);
377. 2              BADDATAfilecount++;
378. 2              return 0;
379. 2          }
380. 2          if (CSTATUS_BADDATA & cm.ce_status)
381. 2          {
382. 2              char pathbuf[PB_MAXPATHLEN];
383. 2              char *p = pathbuf;
384. 2
385. 2              if (getpath(tmp, &pp) :
386. 2                  re_log_stats(
387. 2                      0, "the file %s has BADDATA and was marked", &pp);
388. 2              BADDATAfilecount++;
389. 2              return 0;
390. 2          }
391. 2          if (CSTATUS_BADDATA & cm.ce_status)
392. 2          {
393. 2              char pathbuf[PB_MAXPATHLEN];
394. 2              char *p = pathbuf;
395. 2
396. 2              if (getpath(tmp, &pp) :
397. 2                  re_log_stats(
398. 2                      0, "the file %s has BADDATA and was marked", &pp);
399. 2              BADDATAfilecount++;
400. 2              return 0;
401. 2          }
402. 2          if (CSTATUS_BADDATA & cm.ce_status)
403. 2          {
404. 2              char pathbuf[PB_MAXPATHLEN];
405. 2              char *p = pathbuf;
406. 2
407. 2              if (getpath(tmp, &pp) :
408. 2                  re_log_stats(
409. 2                      0, "the file %s has BADDATA and was marked", &pp);
410. 2              BADDATAfilecount++;
411. 2              return 0;
412. 2          }
413. 2          if (CSTATUS_BADDATA & cm.ce_status)
414. 2          {
415. 2              char pathbuf[PB_MAXPATHLEN];
416. 2              char *p = pathbuf;
417. 2
418. 2              if (getpath(tmp, &pp) :
419. 2                  re_log_stats(
420. 2                      0, "the file %s has BADDATA and was marked", &pp);
421. 2              BADDATAfilecount++;
422. 2              return 0;
423. 2          }
424. 2          if (CSTATUS_BADDATA & cm.ce_status)
425. 2          {
426. 2              char pathbuf[PB_MAXPATHLEN];
427. 2              char *p = pathbuf;
428. 2
429. 2              if (getpath(tmp, &pp) :
430. 2                  re_log_stats(
431. 2                      0, "the file %s has BADDATA and was marked", &pp);
432. 2              BADDATAfilecount++;
433. 2              return 0;
434. 2          }
435. 2          if (CSTATUS_BADDATA & cm.ce_status)
436. 2          {
437. 2              char pathbuf[PB_MAXPATHLEN];
438. 2              char *p = pathbuf;
439. 2
440. 2              if (getpath(tmp, &pp) :
441. 2                  re_log_stats(
442. 2                      0, "the file %s has BADDATA and was marked", &pp);
443. 2              BADDATAfilecount++;
444. 2              return 0;
445. 2          }
446. 2          if (CSTATUS_BADDATA & cm.ce_status)
447. 2          {
448. 2              char pathbuf[PB_MAXPATHLEN];
449. 2              char *p = pathbuf;
450. 2
451. 2              if (getpath(tmp, &pp) :
452. 2                  re_log_stats(
453. 2                      0, "the file %s has BADDATA and was marked", &pp);
454. 2              BADDATAfilecount++;
455. 2              return 0;
456. 2          }
457. 2          if (CSTATUS_BADDATA & cm.ce_status)
458. 2          {
459. 2              char pathbuf[PB_MAXPATHLEN];
460. 2              char *p = pathbuf;
461. 2
462. 2              if (getpath(tmp, &pp) :
463. 2                  re_log_stats(
464. 2                      0, "the file %s has BADDATA and was marked", &pp);
465. 2              BADDATAfilecount++;
466. 2              return 0;
467. 2          }
468. 2          if (CSTATUS_BADDATA & cm.ce_status)
469. 2          {
470. 2              char pathbuf[PB_MAXPATHLEN];
471. 2              char *p = pathbuf;
472. 2
473. 2              if (getpath(tmp, &pp) :
474. 2                  re_log_stats(
475. 2                      0, "the file %s has BADDATA and was marked", &pp);
476. 2              BADDATAfilecount++;
477. 2              return 0;
478. 2          }
479. 2          if (CSTATUS_BADDATA & cm.ce_status)
480. 2          {
481. 2              char pathbuf[PB_MAXPATHLEN];
482. 2              char *p = pathbuf;
483. 2
484. 2              if (getpath(tmp, &pp) :
485. 2                  re_log_stats(
486. 2                      0, "the file %s has BADDATA and was marked", &pp);
487. 2              BADDATAfilecount++;
488. 2              return 0;
489. 2          }
490. 2          if (CSTATUS_BADDATA & cm.ce_status)
491. 2          {
492. 2              char pathbuf[PB_MAXPATHLEN];
493. 2              char *p = pathbuf;
494. 2
495. 2              if (getpath(tmp, &pp) :
496. 2                  re_log_stats(
497. 2                      0, "the file %s has BADDATA and was marked", &pp);
498. 2              BADDATAfilecount++;
499. 2              return 0;
500. 2          }
501. 2          if (CSTATUS_BADDATA & cm.ce_status)
502. 2          {
503. 2              char pathbuf[PB_MAXPATHLEN];
504. 2              char *p = pathbuf;
505. 2
506. 2              if (getpath(tmp, &pp) :
507. 2                  re_log_stats(
508. 2                      0, "the file %s has BADDATA and was marked", &pp);
509. 2              BADDATAfilecount++;
510. 2              return 0;
511. 2          }
512. 2          if (CSTATUS_BADDATA & cm.ce_status)
513. 2          {
514. 2              char pathbuf[PB_MAXPATHLEN];
515. 2              char *p = pathbuf;
516. 2
517. 2              if (getpath(tmp, &pp) :
518. 2                  re_log_stats(
519. 2                      0, "the file %s has BADDATA and was marked", &pp);
520. 2              BADDATAfilecount++;
521. 2              return 0;
522. 2          }
523. 2          if (CSTATUS_BADDATA & cm.ce_status)
524. 2          {
525. 2              char pathbuf[PB_MAXPATHLEN];
526. 2              char *p = pathbuf;
527. 2
528. 2              if (getpath(tmp, &pp) :
529. 2                  re_log_stats(
530. 2                      0, "the file %s has BADDATA and was marked", &pp);
531. 2              BADDATAfilecount++;
532. 2              return 0;
533. 2          }
534. 2          if (CSTATUS_BADDATA & cm.ce_status)
535. 2          {
536. 2              char pathbuf[PB_MAXPATHLEN];
537. 2              char *p = pathbuf;
538. 2
539. 2              if (getpath(tmp, &pp) :
540. 2                  re_log_stats(
541. 2                      0, "the file %s has BADDATA and was marked", &pp);
542. 2              BADDATAfilecount++;
543. 2              return 0;
544. 2          }
545. 2          if (CSTATUS_BADDATA & cm.ce_status)
546. 2          {
547. 2              char pathbuf[PB_MAXPATHLEN];
548. 2              char *p = pathbuf;
549. 2
550. 2              if (getpath(tmp, &pp) :
551. 2                  re_log_stats(
552. 2                      0, "the file %s has BADDATA and was marked", &pp);
553. 2              BADDATAfilecount++;
554. 2              return 0;
555. 2          }
556. 2          if (CSTATUS_BADDATA & cm.ce_status)
557. 2          {
558. 2              char pathbuf[PB_MAXPATHLEN];
559. 2              char *p = pathbuf;
560. 2
561. 2              if (getpath(tmp, &pp) :
562. 2                  re_log_stats(
563. 2                      0, "the file %s has BADDATA and was marked", &pp);
564. 2              BADDATAfilecount++;
565. 2              return 0;
566. 2          }
567. 2          if (CSTATUS_BADDATA & cm.ce_status)
568. 2          {
569. 2              char pathbuf[PB_MAXPATHLEN];
570. 2              char *p = pathbuf;
571. 2
572. 2              if (getpath(tmp, &pp) :
573. 2                  re_log_stats(
574. 2                      0, "the file %s has BADDATA and was marked", &pp);
575. 2              BADDATAfilecount++;
576. 2              return 0;
577. 2          }
578. 2          if (CSTATUS_BADDATA & cm.ce_status)
579. 2          {
580. 2              char pathbuf[PB_MAXPATHLEN];
581. 2              char *p = pathbuf;
582. 2
583. 2              if (getpath(tmp, &pp) :
584. 2                  re_log_stats(
585. 2                      0, "the file %s has BADDATA and was marked", &pp);
586. 2              BADDATAfilecount++;
587. 2              return 0;
588. 2          }
589. 2          if (CSTATUS_BADDATA & cm
```

```

394 2      mcp->LmHis++;
395 2      /*
396 2      * To avoid double logging on mark, do not log BADDATA files.
397 2      */
398 2
400 2      if (! CESTAT_BADDATA & ctm-ce.status)
401 3      {
402 3          Markmarkkbhloglogecho(tmp);
403 2      }
404 2
405 2      /*
406 2      * If this is a DS_NONE record and has no catlm,
407 2      * then we need to remember where we can find the
408 2      * actual catlm information for this node.
409 2      */
410 2      if (tmp->Lm_catlm == -1)
411 3      {
412 3          eerror_ly "rel_status";
413 3
414 3          if (E_SUCCESS != (rel_status = add_denom(ctx, tmp)))
415 4          {
416 4              return rel_status;
417 4          }
418 4
419 4          /*
420 4          * If we're marking lets add the bfile to the volid list.
421 4          * If needed adding the volume to the list and incrementing
422 4          * the bfile dependency count.
423 4          */
424 2      }
425 2
426 2      rcx->ebvolid = ebvolid_add_bfile_to_volid_count {
427 3          (ebfa_and_ly*) & {
428 3              ctm-ce.bfileid,
429 3              kcp.status);
430 3
431 3          /*
432 3          * This error does not effect the recover,
433 3          * but lets log it
434 3          */
435 3
436 3          the_internal_error { ep.status,
437 3              *NOTE: in ebvolid_add_bfile_to_volid_count {
438 3                  eb_obsidascia { add_ce_bfileid, NULL,
439 3                      km.ce.bfileid, NULL,
440 3                      km.EBRTSIDASCIT_WITH_DOTS } );
441 3              }
442 3
443 3          /*
444 3          * If the summary is valid, update it for this bfile.
445 3          */
446 2      }
447 2      if (rcx->rc.mark_summary_valid)
448 3      {
449 3          add_to_summary(&rcx->rc.mark_summary, &ctm);
450 2      }
451 2
452 2      /*
453 2      * If we have reached the boundary for reporting progress
454 2      * prior to marking this file, call the progress routine
455 2      * and verify it is OK to continue
456 2
457 2      FSNmarknumic10      Page 62 of 248

```


File Jan 04 16:35:25 2008	unmark_tree	Page 66 of 246	File Jan 04 16:35:25 2008	unmark_tree	Page 66 of 246
595 1 596 1 597 1 598 1 599 1 600 2 601 2 602 2 603 2 604 2 605 2 606 2 607 2 608 2 610 2 611 2 612 2 613 2 615 2	<pre> /* * unmark this guy */ if (MARKED(rcx->rc_marks, tmp)) { if (&rcm_cim) rcac_atm(cim); /* * get catalog record */ rcac_getcatm(rcx->rc_mcp, tmp, &cim); /* * Skip file if we're only unmarking bad files * and this one is NOT bad. */ if (mcp->no_badfiles && ! (CSTATUS_BADDATA & cim.ce_status)) { goto unmark_children; } /* * Count bad files */ if (CSTATUS_BADDATA & cim.ce_status) { BADDATAFilemarkCount++; } /* * Log if this is an unmark BADDATA files only unmark call */ if (mcp->no_badfiles) { char pathbuf[PATH_MAX+1]; char *pnp = pathbuf; to_getpath(tmp, &pnp); the_log_stats(0, "The BADDATA file %s was unmarked", pnp); } /* * 1) Clearing mark this bitfile for recovery bit. * 2) decrementing marks for the plane * 3) decrementing total marks * 4) incrementing the unmarked bitfile count for call to unmark_tree */ TCUNMARK(rcx->rc_marks, tmp); rcx->rc_marks_by_plane[tmp->trm_plane]--; rcx->rc_marks_total--; mcp->n_this++; </pre>		615 2 616 2 617 2 618 2 620 2 621 3 622 3 623 2 625 2 626 2 627 2 628 2 630 2 631 2 632 2 633 3 634 3 635 3 636 3 638 3 639 3 640 3 641 3 642 2 644 2 645 2 646 2 647 2 648 2 649 3 650 3 651 2 653 2 654 3 655 2	<pre> /* * To avoid double logging on mark, do not log BADDATA files. * If badfile only has been set. */ if (! (mcp->no_badfiles) && (CSTATUS_BADDATA & cim.ce_status)) { MarkUnmarkedBadLogEcho(tmp); } /* * If we're unmarking let's decrement the file to the void * file dependency count. */ ep_status = bvt_dect_file_to_void_count(rcx->epvlist, (bvt_ujdy), & cim.ce_bitfileid); if (E_SUCCESS != ep_status) { /* * This error does not effect the recover, * but lets log it */ the_internal_error(ep_status, "NOTE: in bvt_dect_file_to_void count{ eb_abbrevidasci(cim.ce_bitfileid, NULL, EB_abbrevidasci(MTR_DOTS)); }); /* * If the summary is valid, update it for this bitfile. */ if (rcx->rc_mark_summary_valid) { sub_from_summary(&rcx->rc_mark_summary, &cim); } /* * undo denote info saved, if any */ if (tmp->trm_cim == -1) { remove_denote(rcx, tmp); } /* * If we have reached the boundary for reporting progress * prior to unmarking this file, call the progress routine * and verify it is OK to continue */ if (((global_start_marks - rcx->rc_marks_total) & 0x3fff) == 0) { /* Don't bother on 0 */ if (global_start_marks - rcx->rc_marks_total > 0) { /* If callback returns non-zero, * flag this operation as cancelled */ if (global_progress_cb(global_start_marks - rcx->rc_marks_total)) </pre>	615 2 616 2 617 2 618 2 620 2 621 3 622 3 623 2 625 2 626 2 627 2 628 2 630 2 631 2 632 2 633 3 634 3 635 3 636 3 638 3 639 3 640 3 641 3 642 2 644 2 645 2 646 2 647 2 648 2 649 3 650 3 651 2 653 2 654 3 655 2
File Jan 04 16:35:25 2008	RSUnmarkum.c 13	Page 65 of 246	File Jan 04 16:35:25 2008	RSUnmarkum.c 14	Page 66 of 246

Page 67 of 248	unmark_tree	Fri Jan 04 16:35:25 2008
675 5	{	
676 5	/* Set the cancelled flag */	
677 5	global_wm_cancelled = TRUE;	
678 5	/* Get out, since user cancelled */	
679 5	return (0);	
680 5	}	
681 4	}	
682 3	}	
683 2	}	
684 1	}	
685 1	unmark_children:	
686 1	if (tmp->en_flags & TMP_KNOWN_NOCHILDREN)	
687 2	{	
688 2	return 0;	
689 1	}	
690 1	if (tmp->no_descend)	
691 1	{	
692 2	(void)matl_apply_children(rcx->rc_mcp, tmp, unmk, {	char *)rcx;
693 1	}	
694 2	return 0;	
695 2	/* end of unmark_tree() */	
696 1	}	
697 1	}	
698 1	}	
Page 68 of 248	check_parent_perms	Fri Jan 04 16:35:25 2008
700	static int	
701	check_parent_perms(struct recover_context *rcx,	
702	char *name, if_known,	
703	char *errorbuff)	
704 1	{	
705 1	fbcat_elem_t ctmp;	
706 1	fbcat_elem_t sctmp = scdm;	
707 1	fbcat_node parent_path[PATN_MAX], *tmptr;	
708 1	char	
709 1	int	
710 1	ok = 1;	
711 2	if (rcx->rc_effective_uid == 0)	
712 1	{	
713 2	return 0;	
714 2	}	
715 1	strcpy(parent_path, name, if_known);	
717 1	if ((tmptr = strchr(parent_path, '/')) == NULL)	
719 1	{	
720 2	return 0;	
721 2	}	
722 1	do	
723 1	{	
724 1	if (tmptr == parent_path)	
725 2	{	
726 2	return 0;	
727 3	}	
728 3	tmptr = strchr(tmptr, '/');	
729 2	if (tmptr == parent_path)	
731 2	{	
732 2	tmp = matl_lookup_path(rcx->rc_mcp, parent_path);	
733 2	/*	
734 2	* get catalog record	
735 2	*/	
736 2	if (tmp != NULL)	
737 2	{	
738 2	matl_getcatm(rcx->rc_mcp, tmp, scdm);	
739 2	if (rcx->rc_effective_uid != cdm.ce_owner)	
740 3	{	
741 3	ok = rcx_permchk(rcx, PERMCHK_R PERMCHK_X, tmp, scdm);	
742 3	}	
743 3	while (ok && (tmptr = strchr(parent_path, '/')) != NULL);	
744 4	if (ok == 0 && errorbuff != NULL)	
745 3	{	
746 3	strcpy(errorbuff, parent_path);	
747 3	}	
748 1	return 1;	
749 1	/*	
750 1	* check_parent_perms */	
751 1	}	
752 1	}	
753 1	}	
754 1	}	
755 1	}	
756 1	}	
757 1	}	

```

160 /*****
161  * RSTSL_ManObject()
162  *
163  * The ManObject operation takes a restorableObject and marks it,
164  * possibly its descendant files for restore based on the input
165  * criteria.
166  * The RSTSL_ManObject call is an asynchronously executed operation
167  * in the Restore Engine that performs the marking.
168  * It stores its results
169  * in buffers supplied by the caller,
170  * so that the synchronous RRC thread can
171  * test for and retrieve results without another Restore Service
172  * call.
173  * Parameters:
174  *   * ThisObject (I) - The restoral object;
175  *   *   can be a leaf object (e.g. a
176  *   *   file), or a container object (
177  *   *   e.g., a directory).
178  *   * time (I) - (
179  *   *   optional) the backup time to perform the mark on --
180  *   *   If not specified,
181  *   *   uses currently selected backup; if
182  *   *   specified,
183  *   *   leaves selected backup time unchanged
184  *   * allowedFiles (I) - allows marking of files of state BADDATA.
185  *   *   descand (I) - Should mark operation descend to operate on the contents
186  *   *   of container objects
187  *   * BadFileCount (O) - returns the file count with BADDATA.
188  *   * PermByFileCount (O) -- returns the file count with permission denied.
189  *   * fileMarked (O) -- returns the file count with permission denied.
190  *   *   O) - return the total files marked after this mark occurred.
191  *   *   O) - return the total directories marked after this mark
192  *   *   occurred.
193  *   * otherMarked (I) - return the total "other" files marked after this mark.
194  *   *   O) - return the total "other" files marked after this mark.
195  *   * progressCB (I) - pointer to callback function to report progress and
196  *   *   test for cancellation
197  *   *   *****
198  *   *****
199  *   *****
200  *   *****
201  *   *****
202  *   *****
203  *   *****
204  *   *****
205  *   *****
206  *   *****
207  *   *****
208  *   *****
209  *   *****
210  *   *****
211  *   *****
212  *   *****
213  *   *****
214  *   *****
215  *   *****
216  *   *****
217  *   *****
218  *   *****
219  *   *****
220  *   *****
221  *   *****
222  *   *****
223  *   *****
224  *   *****
225  *   *****
226  *   *****
227  *   *****
228  *   *****
229  *   *****
230  *   *****
231  *   *****
232  *   *****
233  *   *****
234  *   *****
235  *   *****
236  *   *****
237  *   *****
238  *   *****
239  *   *****
240  *   *****
241  *   *****
242  *   *****
243  *   *****
244  *   *****
245  *   *****
246  *   *****
247  *   *****
248  *   *****
249  *   *****
250  *   *****
251  *   *****
252  *   *****
253  *   *****
254  *   *****
255  *   *****
256  *   *****
257  *   *****
258  *   *****
259  *   *****
260  *   *****
261  *   *****
262  *   *****
263  *   *****
264  *   *****
265  *   *****
266  *   *****
267  *   *****
268  *   *****
269  *   *****
270  *   *****
271  *   *****
272  *   *****
273  *   *****
274  *   *****
275  *   *****
276  *   *****
277  *   *****
278  *   *****
279  *   *****
280  *   *****
281  *   *****
282  *   *****
283  *   *****
284  *   *****
285  *   *****
286  *   *****
287  *   *****
288  *   *****
289  *   *****
290  *   *****
291  *   *****
292  *   *****
293  *   *****
294  *   *****
295  *   *****
296  *   *****
297  *   *****
298  *   *****
299  *   *****
300  *   *****
301  *   *****
302  *   *****
303  *   *****
304  *   *****
305  *   *****
306  *   *****
307  *   *****
308  *   *****
309  *   *****
310  *   *****
311  *   *****
312  *   *****
313  *   *****
314  *   *****
315  *   *****
316  *   *****
317  *   *****
318  *   *****
319  *   *****
320  *   *****
321  *   *****
322  *   *****
323  *   *****
324  *   *****
325  *   *****
326  *   *****
327  *   *****
328  *   *****
329  *   *****
330  *   *****
331  *   *****
332  *   *****
333  *   *****
334  *   *****
335  *   *****
336  *   *****
337  *   *****
338  *   *****
339  *   *****
340  *   *****
341  *   *****
342  *   *****
343  *   *****
344  *   *****
345  *   *****
346  *   *****
347  *   *****
348  *   *****
349  *   *****
350  *   *****
351  *   *****
352  *   *****
353  *   *****
354  *   *****
355  *   *****
356  *   *****
357  *   *****
358  *   *****
359  *   *****
360  *   *****
361  *   *****
362  *   *****
363  *   *****
364  *   *****
365  *   *****
366  *   *****
367  *   *****
368  *   *****
369  *   *****
370  *   *****
371  *   *****
372  *   *****
373  *   *****
374  *   *****
375  *   *****
376  *   *****
377  *   *****
378  *   *****
379  *   *****
380  *   *****
381  *   *****
382  *   *****
383  *   *****
384  *   *****
385  *   *****
386  *   *****
387  *   *****
388  *   *****
389  *   *****
390  *   *****
391  *   *****
392  *   *****
393  *   *****
394  *   *****
395  *   *****
396  *   *****
397  *   *****
398  *   *****
399  *   *****
400  *   *****
401  *   *****
402  *   *****
403  *   *****
404  *   *****
405  *   *****
406  *   *****
407  *   *****
408  *   *****
409  *   *****
410  *   *****
411  *   *****
412  *   *****
413  *   *****
414  *   *****
415  *   *****
416  *   *****
417  *   *****
418  *   *****
419  *   *****
420  *   *****
421  *   *****
422  *   *****
423  *   *****
424  *   *****
425  *   *****
426  *   *****
427  *   *****
428  *   *****
429  *   *****
430  *   *****
431  *   *****
432  *   *****
433  *   *****
434  *   *****
435  *   *****
436  *   *****
437  *   *****
438  *   *****
439  *   *****
440  *   *****
441  *   *****
442  *   *****
443  *   *****
444  *   *****
445  *   *****
446  *   *****
447  *   *****
448  *   *****
449  *   *****
450  *   *****
451  *   *****
452  *   *****
453  *   *****
454  *   *****
455  *   *****
456  *   *****
457  *   *****
458  *   *****
459  *   *****
460  *   *****
461  *   *****
462  *   *****
463  *   *****
464  *   *****
465  *   *****
466  *   *****
467  *   *****
468  *   *****
469  *   *****
470  *   *****
471  *   *****
472  *   *****
473  *   *****
474  *   *****
475  *   *****
476  *   *****
477  *   *****
478  *   *****
479  *   *****
480  *   *****
481  *   *****
482  *   *****
483  *   *****
484  *   *****
485  *   *****
486  *   *****
487  *   *****
488  *   *****
489  *   *****
490  *   *****
491  *   *****
492  *   *****
493  *   *****
494  *   *****
495  *   *****
496  *   *****
497  *   *****
498  *   *****
499  *   *****
500  *   *****
501  *   *****
502  *   *****
503  *   *****
504  *   *****
505  *   *****
506  *   *****
507  *   *****
508  *   *****
509  *   *****
510  *   *****
511  *   *****
512  *   *****
513  *   *****
514  *   *****
515  *   *****
516  *   *****
517  *   *****
518  *   *****
519  *   *****
520  *   *****
521  *   *****
522  *   *****
523  *   *****
524  *   *****
525  *   *****
526  *   *****
527  *   *****
528  *   *****
529  *   *****
530  *   *****
531  *   *****
532  *   *****
533  *   *****
534  *   *****
535  *   *****
536  *   *****
537  *   *****
538  *   *****
539  *   *****
540  *   *****
541  *   *****
542  *   *****
543  *   *****
544  *   *****
545  *   *****
546  *   *****
547  *   *****
548  *   *****
549  *   *****
550  *   *****
551  *   *****
552  *   *****
553  *   *****
554  *   *****
555  *   *****
556  *   *****
557  *   *****
558  *   *****
559  *   *****
560  *   *****
561  *   *****
562  *   *****
563  *   *****
564  *   *****
565  *   *****
566  *   *****
567  *   *****
568  *   *****
569  *   *****
570  *   *****
571  *   *****
572  *   *****
573  *   *****
574  *   *****
575  *   *****
576  *   *****
577  *   *****
578  *   *****
579  *   *****
580  *   *****
581  *   *****
582  *   *****
583  *   *****
584  *   *****
585  *   *****
586  *   *****
587  *   *****
588  *   *****
589  *   *****
590  *   *****
591  *   *****
592  *   *****
593  *   *****
594  *   *****
595  *   *****
596  *   *****
597  *   *****
598  *   *****
599  *   *****
600  *   *****
601  *   *****
602  *   *****
603  *   *****
604  *   *****
605  *   *****
606  *   *****
607  *   *****
608  *   *****
609  *   *****
610  *   *****
611  *   *****
612  *   *****
613  *   *****
614  *   *****
615  *   *****
616  *   *****
617  *   *****
618  *   *****
619  *   *****
620  *   *****
621  *   *****
622  *   *****
623  *   *****
624  *   *****
625  *   *****
626  *   *****
627  *   *****
628  *   *****
629  *   *****
630  *   *****
631  *   *****
632  *   *****
633  *   *****
634  *   *****
635  *   *****
636  *   *****
637  *   *****
638  *   *****
639  *   *****
640  *   *****
641  *   *****
642  *   *****
643  *   *****
644  *   *****
645  *   *****
646  *   *****
647  *   *****
648  *   *****
649  *   *****
650  *   *****
651  *   *****
652  *   *****
653  *   *****
654  *   *****
655  *   *****
656  *   *****
657  *   *****
658  *   *****
659  *   *****
660  *   *****
661  *   *****
662  *   *****
663  *   *****
664  *   *****
665  *   *****
666  *   *****
667  *   *****
668  *   *****
669  *   *****
670  *   *****
671  *   *****
672  *   *****
673  *   *****
674  *   *****
675  *   *****
676  *   *****
677  *   *****
678  *   *****
679  *   *****
680  *   *****
681  *   *****
682  *   *****
683  *   *****
684  *   *****
685  *   *****
686  *   *****
687  *   *****
688  *   *****
689  *   *****
690  *   *****
691  *   *****
692  *   *****
693  *   *****
694  *   *****
695  *   *****
696  *   *****
697  *   *****
698  *   *****
699  *   *****
700  *   *****
701  *   *****
702  *   *****
703  *   *****
704  *   *****
705  *   *****
706  *   *****
707  *   *****
708  *   *****
709  *   *****
710  *   *****
711  *   *****
712  *   *****
713  *   *****
714  *   *****
715  *   *****
716  *   *****
717  *   *****
718  *   *****
719  *   *****
720  *   *****
721  *   *****
722  *   *****
723  *   *****
724  *   *****
725  *   *****
726  *   *****
727  *   *****
728  *   *****
729  *   *****
730  *   *****
731  *   *****
732  *   *****
733  *   *****
734  *   *****
735  *   *****
736  *   *****
737  *   *****
738  *   *****
739  *   *****
740  *   *****
741  *   *****
742  *   *****
743  *   *****
744  *   *****
745  *   *****
746  *   *****
747  *   *****
748  *   *****
749  *   *****
750  *   *****
751  *   *****
752  *   *****
753  *   *****
754  *   *****
755  *   *****
756  *   *****
757  *   *****
758  *   *****
759  *   *****
760  *   *****
761  *   *****
762  *   *****
763  *   *****
764  *   *****
765  *   *****
766  *   *****
767  *   *****
768  *   *****
769  *   *****
770  *   *****
771  *   *****
772  *   *****
773  *   *****
774  *   *****
775  *   *****
776  *   *****
777  *   *****
778  *   *****
779  *   *****
780  *   *****
781  *   *****
782  *   *****
783  *   *****
784  *   *****
785  *   *****
786  *   *****
787  *   *****
788  *   *****
789  *   *****
790  *   *****
791  *   *****
792  *   *****
793  *   *****
794  *   *****
795  *   *****
796  *   *****
797  *   *****
798  *   *****
799  *   *****
800  *   *****
801  *   *****
802  *   *****
803  *   *****
804  *   *****
805  *   *****
806  *   *****
807  *   *****
808  *   *****
809  *   *****
810  *   *****
811  *   *****
812  *   *****
813  *   *****
814  *   *****
815  *   *****
816  *   *****
817  *   *****
818  *   *****
819  *   *****
820  *   *****
821  *   *****
822  *   *****
823  *   *****
824  *   *****
825  *   *****
826  *   *****
827  *   *****
828  *   *****
829  *   *****
830  *   *****
831  *   *****
832  *   *****
833  *   *****
834  *   *****
835  *   *****
836  *   *****
837  *   *****
838  *   *****
839  *   *****
840  *   *****
841  *   *****
842  *   *****
843  *   *****
844  *   *****
845  *   *****
846  *   *****
847  *   *****
848  *   *****
849  *   *****
850  *   *****
851  *   *****
852  *   *****
853  *   *****
854  *   *****
855  *   *****
856  *   *****
857  *   *****
858  *   *****
859  *   *****
860  *   *****
861  *   *****
862  *   *****
863  *   *****
864  *   *****
865  *   *****
866  *   *****
867  *   *****
868  *   *****
869  *   *****
870  *   *****
871  *   *****
872  *   *****
873  *   *****
874  *   *****
875  *   *****
876  *   *****
877  *   *****
878  *   *****
879  *   *****
880  *   *****
881  *   *****
882  *   *****
883  *   *****
884  *   *****
885  *   *****
886  *   *****
887  *   *****
888  *   *****
889  *   *****
890  *   *****
891  *   *****
892  *   *****
893  *   *****
894  *   *****
895  *   *****
896  *   *****
897  *   *****
898  *   *****
899  *   *****
900  *   *****
901  *   *****
902  *   *****
903  *   *****
904  *   *****
905  *   *****
906  *   *****
907  *   *****
908  *   *****
909  *   *****
910  *   *****
911  *   *****
912  *   *****
913  *   *****
914  *   *****
915  *   *****
916  *   *****
917  *   *****
918  *   *****
919  *   *****
920  *   *****
921  *   *****
922  *   *****
923  *   *****
924  *   *****
925  *   *****
926  *   *****
927  *   *****
928  *   *****
929  *   *****
930  *   *****
931  *   *****
932  *   *****
933  *   *****
934  *   *****
935  *   *****
936  *   *****
937  *   *****
938  *   *****
939  *   *****
940  *   *****
941  *   *****
942  *   *****
943  *   *****
944  *   *****
945  *   *****
946  *   *****
947  *   *****
948  *   *****
949  *   *****
950  *   *****
951  *   *****
952  *   *****
953  *   *****
954  *   *****
955  *   *****
956  *   *****
957  *   *****
958  *   *****
959  *   *****
960  *   *****
961  *   *****
962  *   *****
963  *   *****
964  *   *****
965  *   *****
966  *   *****
967  *   *****
968  *   *****
969  *   *****
970  *   *****
971  *   *****
972  *   *****
973  *   *****
974  *   *****
975  *   *****
976  *   *****
977  *   *****
978  *   *****
979  *   *****
980  *   *****
981  *   *****
982  *   *****
983  *   *****
984  *   *****
985  *   *****
986  *   *****
987  *   *****
988  *   *****
989  *   *****
990  *   *****
991  *   *****
992  *   *****
993  *   *****
994  *   *****
995  *   *****
996  *   *****
997  *   *****
998  *   *****
999  *   *****
1000  *   *****

```

```

808 1 time_t save_time = 0;
809 1 backup_time = 0;
810 1 RSTSL_ManObject *thisObject;
811 1 backup_flags = 0;
812 1
813 1
814 1
815 1
816 1
817 1
818 1
819 1
820 1
821 1
822 1
823 1
824 1
825 1
826 1
827 1
828 1
829 1
830 1
831 1
832 1
833 1
834 1
835 1
836 1
837 1
838 1
839 1
840 1
841 1
842 1
843 1
844 1
845 1
846 1
847 1
848 1
849 1
850 1
851 1
852 1
853 1
854 1
855 1
856 1
857 1
858 1
859 1
860 1
861 1
862 1
863 1
864 1
865 1
866 1
867 1
868 1
869 1
870 1
871 1
872 1
873 1
874 1
875 1
876 1
877 1
878 1
879 1
880 1
881 1
882 1
883 1
884 1
885 1
886 1
887 1
888 1
889 1
890 1
891 1
892 1
893 1
894 1
895 1
896 1
897 1
898 1
899 1
900 1
901 1
902 1
903 1
904 1
905 1
906 1
907 1
908 1
909 1
910 1
911 1
912 1
913 1
914 1
915 1
916 1
917 1
918 1
919 1
920 1
921 1
922 1
923 1
924 1
925 1
926 1
927 1
928 1
929 1
930 1
931 1
932 1
933 1
934 1
935 1
936 1
937 1
938 1
939 1
940 1
941 1
942 1
943 1
944 1
945 1
946 1
947 1
948 1
949 1
950 1
951 1
952 1
953 1
954 1
955 1
956 1
957 1
958 1
959 1
960 1
961 1
962 1
963 1
964 1
965 1
966 1
967 1
968 1
969 1
970 1
971 1
972 1
973 1
974 1
975 1
976 1
977 1
978 1
979 1
980 1
981 1
982 1
983 1
984 1
985 1
986 1
987 1
988 1
989 1
990 1
991 1
992 1
993 1
994 1
995 1
996 1
997 1
998 1
999 1
1000 1

```


Page 71 of 248	NSTL_MarkObject	Fri Jan 04 16:35:25 2008
970 3 971 1 972 3 973 3 974 3 975 3 976 1 977 2 978 2 979 2 980 1	<pre> rtc = tcp->currentPiptr-> pFuncIndex[pFuncIndexBackupTime]) else { rtc = NSTL_descCurrentBackupTime(leave_time); if (rtc == E_SUCCESS) rtc = NSTL_descBackupTime(backup_time, backup_flags); } if (rtc != E_SUCCESS) return rtc; /* If this isn't a network backup restore, call plugin to do mark */ if (tcp->rc_backup_app != 0) { rtc = tcp->currentPiptr->pFuncIndexMark] { tcp, thisobject, allowBadfiles, descend, BadFilesCount, PermBadFilesCount, allowMarkedFiles, disMarked, otherMarked, progressCB); /* save mark summary in restore context */ tcp->rc_mark_summary.nfiles = fileMarked; tcp->rc_mark_summary.ndirs = dirMarked; tcp->rc_mark_summary.nother = otherMarked; tcp->rc_mark_summary.len_md_files_high = lenMarkedFiles_high; tcp->rc_mark_summary.len_md_files_low = lenMarkedFiles_low; if (save_time) tcp->currentPiptr->pFuncIndex[pFuncIndexBackupTime] { tcp, save_time, backup_flags); return rtc; } /* Initial global marking/unmarking cancel flag to FALSE */ global = was_cancelled = FALSE; global_progress.cb = progressCB; global_start_marks = tcp->rc_marks.total; mc.n.this = 0; mc.n.before = tcp->rc_marks.total; /* * BadFilesCount = *PermBadFilesCount = 0; */ /* * The Below are static variables global to this module * api_markumark.c * They get initialized in EXNST_MarkObject to 0. * They get incremented in mark_tree() * when the conditions of mark are * encountered. * They get assigned to the output parameters to EXNST_MarkObject * of user * the mark tree call. </pre>	<pre> 974 1 976 1 978 1 979 1 980 1 981 1 982 1 983 2 984 2 985 2 986 2 987 1 988 1 989 2 990 2 991 1 992 1 993 1 994 1 995 1 996 1 997 1 998 1 999 1 1000 1 </pre>
Page 72 of 248	NSTL_MarkObject	Fri Jan 04 16:35:25 2008
994 1 996 1 998 1 999 1 1000 1	<pre> /* * BADDATAFilesCount = PermBadFilesCount = 0; * tcp->rc_md_context = (char *)mc; /* * Input parameters * 1) allowedfiles -- does marking of Badfiles * 2) descend -- does mark descend into the contents of * directories. */ if (allowBadfiles) { mc.no_badfiles = TRUE; } else { mc.no_badfiles = FALSE; } if (descend) { mc.no_descend = TRUE; } else { mc.no_descend = FALSE; } /* * Open up the saveasec db during mark command. */ rtc = ss_open_saveasec_db(); if (0 != rtc) { return (BP_RB_RECOVER_CANT_OPEN_S3DB); } /* * ssdb opened was used to indicate whether or not * ss_open_saveasec_db was successful. It was set to TRUE * when ss_open_saveasec_db() succeeded, so we'll remember * to call ss_close_saveasec_db later. This is no longer * necessary because we would have returned an error if * ss_open_saveasec_db failed. It's left in to minimize * code changes. */ ssdb_opened = TRUE; /* * validate restorable object with the current mcot context */ /* * Initialize tmp_chisobject to NULL so that if * mcot_lookup_path * does not set it to NULL in the case of failure, the code * can work correctly. */ tree_node *tmp_chisobject = NULL; </pre>	<pre> 994 1 996 1 998 1 999 1 1000 1 </pre>
Page 72 of 248	NSumarmumc20	Fri Jan 04 16:35:25 2008

```

998 2      tmp_chisbobject = mcat_lookup_path
999 3      {
1000 4          if (tmp_chisbobject == NULL)
1001 5              /*
1002 6                  * If we opened sashb then lets close it!
1003 7                  *
1004 8                  * restoreable object must be corrupt mcat_lookup_path
1005 9                  *
1006 10                 * Locate the tree node per for files
1007 11                 */
1008 12                 if (sashb_opened)
1009 13                     {
1010 14                         ss_close_saveset_db();
1011 15                     }
1012 16                 return EP_RB_RECOVER_BAD_CONTENT;
1013 17             }
1014 18             if (tmp_chisbobject->in_mplane !=
1015 19                 ((mibackupobj_data *) (
1016 20                     thisbobject->appdata.data))->objinmplane)
1017 21             {
1018 22                 /*
1019 23                     * If we opened sashb then lets close it!
1020 24                     *
1021 25                     * restoreable object must be out of context, return error
1022 26                     */
1023 27                 if (sashb_opened)
1024 28                     {
1025 29                         ss_close_saveset_db();
1026 30                     }
1027 31                 return EP_RB_RECOVER_BAD_CONTENT;
1028 32             }
1029 33             /*
1030 34                 * If we made it here then thisbobject is valid.
1031 35                 */
1032 36             if (check_parent_perms(
1033 37                 tcp, thisbobject->root.objName, errorbuf))
1034 38             {
1035 39                 rbe_log_stats(0,
1036 40                     "Permission denied for file '%s'\n %s",
1037 41                     thisbobject->root.objName,
1038 42                     "parent directory permissions");
1039 43             }
1040 44             else
1041 45             {
1042 46                 /*
1043 47                     * mark_tree always returns 0
1044 48                     */
1045 49                 (void) mark_tree(
1046 50                     tcp, tmp_chisbobject, thisbobject->root.objName);
1047 51             }
1048 52             /*
1049 53                 * The Below are static variables global to this module
1050 54                 *
1051 55                 * They get initialized in EDMRST_MarkObject.c
1052 56                 * They get incremented in mark_tree()
1053 57                 *
1054 58                 * They get assigned to the output parameters to
1055 59                 *
1056 60                 */
1057 61             RSTSLmarknumc21
1058 62         }
1059 63     }
1060 64     Fri Jan 04 16:35:25 2008
1061 65     RSTSLmarknumc21
1062 66     Page 73 of 248

```

```

1067 1      * the mark_tree call.
1068 2      */
1069 3      *BadFilesCount = BADDATAFilesCount;
1070 4      *PermDeniedCount = PermissionDeniedFilesCount;
1071 5
1072 6      /*
1073 7          * If the summary is valid, get the total number of marked files,
1074 8          * add the number of files that resulted directly from the above mark tree
1075 9          * number of files that resulted directly from the above mark tree
1076 10         */
1077 11         if (tcp->rc.mark_summary_valid)
1078 12             {
1079 13                 *fileMarked = tcp->rc.mark_summary_files;
1080 14                 *otherMarked = tcp->rc.mark_summary_nother;
1081 15             }
1082 16         else
1083 17         {
1084 18             /*
1085 19                 * This error is not fatal, the output variable *Marked,
1086 20                 * will be zero, we should never get this error.
1087 21                 */
1088 22             rbe_log_stats(
1089 23                 0, "Internal error: mark summary not valid in EDMRST_MarkObject()");
1090 24         }
1091 25         /*
1092 26             * If we opened it then lets close it!
1093 27             */
1094 28         if (sashb_opened)
1095 29             {
1096 30                 ss_close_saveset_db();
1097 31             }
1098 32         if (save_time)
1099 33             RSTSL_SetBackupForTime (save_time, backup_flags );
1100 34         return( E_SUCCESS );
1101 35     }
1102 36     /* end of RSTSL_MarkObject () */
1103 37 }
1104 38
1105 39
1106 40
1107 41
1108 42
1109 43
1110 44
1111 45
1112 46
1113 47
1114 48
1115 49
1116 50
1117 51
1118 52
1119 53
1120 54
1121 55
1122 56
1123 57
1124 58
1125 59
1126 60
1127 61
1128 62
1129 63
1130 64
1131 65
1132 66
1133 67
1134 68
1135 69
1136 70
1137 71
1138 72
1139 73
1140 74
1141 75
1142 76
1143 77
1144 78
1145 79
1146 80
1147 81
1148 82
1149 83
1150 84
1151 85
1152 86
1153 87
1154 88
1155 89
1156 90
1157 91
1158 92
1159 93
1160 94
1161 95
1162 96
1163 97
1164 98
1165 99
1166 100
1167 101
1168 102
1169 103
1170 104
1171 105
1172 106
1173 107
1174 108
1175 109
1176 110
1177 111
1178 112
1179 113
1180 114
1181 115
1182 116
1183 117
1184 118
1185 119
1186 120
1187 121
1188 122
1189 123
1190 124
1191 125
1192 126
1193 127
1194 128
1195 129
1196 130
1197 131
1198 132
1199 133
1200 134
1201 135
1202 136
1203 137
1204 138
1205 139
1206 140
1207 141
1208 142
1209 143
1210 144
1211 145
1212 146
1213 147
1214 148
1215 149
1216 150
1217 151
1218 152
1219 153
1220 154
1221 155
1222 156
1223 157
1224 158
1225 159
1226 160
1227 161
1228 162
1229 163
1230 164
1231 165
1232 166
1233 167
1234 168
1235 169
1236 170
1237 171
1238 172
1239 173
1240 174
1241 175
1242 176
1243 177
1244 178
1245 179
1246 180
1247 181
1248 182
1249 183
1250 184
1251 185
1252 186
1253 187
1254 188
1255 189
1256 190
1257 191
1258 192
1259 193
1260 194
1261 195
1262 196
1263 197
1264 198
1265 199
1266 200
1267 201
1268 202
1269 203
1270 204
1271 205
1272 206
1273 207
1274 208
1275 209
1276 210
1277 211
1278 212
1279 213
1280 214
1281 215
1282 216
1283 217
1284 218
1285 219
1286 220
1287 221
1288 222
1289 223
1290 224
1291 225
1292 226
1293 227
1294 228
1295 229
1296 230
1297 231
1298 232
1299 233
1300 234
1301 235
1302 236
1303 237
1304 238
1305 239
1306 240
1307 241
1308 242
1309 243
1310 244
1311 245
1312 246
1313 247
1314 248
1315 249
1316 250
1317 251
1318 252
1319 253
1320 254
1321 255
1322 256
1323 257
1324 258
1325 259
1326 260
1327 261
1328 262
1329 263
1330 264
1331 265
1332 266
1333 267
1334 268
1335 269
1336 270
1337 271
1338 272
1339 273
1340 274
1341 275
1342 276
1343 277
1344 278
1345 279
1346 280
1347 281
1348 282
1349 283
1350 284
1351 285
1352 286
1353 287
1354 288
1355 289
1356 290
1357 291
1358 292
1359 293
1360 294
1361 295
1362 296
1363 297
1364 298
1365 299
1366 300
1367 301
1368 302
1369 303
1370 304
1371 305
1372 306
1373 307
1374 308
1375 309
1376 310
1377 311
1378 312
1379 313
1380 314
1381 315
1382 316
1383 317
1384 318
1385 319
1386 320
1387 321
1388 322
1389 323
1390 324
1391 325
1392 326
1393 327
1394 328
1395 329
1396 330
1397 331
1398 332
1399 333
1400 334
1401 335
1402 336
1403 337
1404 338
1405 339
1406 340
1407 341
1408 342
1409 343
1410 344
1411 345
1412 346
1413 347
1414 348
1415 349
1416 350
1417 351
1418 352
1419 353
1420 354
1421 355
1422 356
1423 357
1424 358
1425 359
1426 360
1427 361
1428 362
1429 363
1430 364
1431 365
1432 366
1433 367
1434 368
1435 369
1436 370
1437 371
1438 372
1439 373
1440 374
1441 375
1442 376
1443 377
1444 378
1445 379
1446 380
1447 381
1448 382
1449 383
1450 384
1451 385
1452 386
1453 387
1454 388
1455 389
1456 390
1457 391
1458 392
1459 393
1460 394
1461 395
1462 396
1463 397
1464 398
1465 399
1466 400
1467 401
1468 402
1469 403
1470 404
1471 405
1472 406
1473 407
1474 408
1475 409
1476 410
1477 411
1478 412
1479 413
1480 414
1481 415
1482 416
1483 417
1484 418
1485 419
1486 420
1487 421
1488 422
1489 423
1490 424
1491 425
1492 426
1493 427
1494 428
1495 429
1496 430
1497 431
1498 432
1499 433
1500 434
1501 435
1502 436
1503 437
1504 438
1505 439
1506 440
1507 441
1508 442
1509 443
1510 444
1511 445
1512 446
1513 447
1514 448
1515 449
1516 450
1517 451
1518 452
1519 453
1520 454
1521 455
1522 456
1523 457
1524 458
1525 459
1526 460
1527 461
1528 462
1529 463
1530 464
1531 465
1532 466
1533 467
1534 468
1535 469
1536 470
1537 471
1538 472
1539 473
1540 474
1541 475
1542 476
1543 477
1544 478
1545 479
1546 480
1547 481
1548 482
1549 483
1550 484
1551 485
1552 486
1553 487
1554 488
1555 489
1556 490
1557 491
1558 492
1559 493
1560 494
1561 495
1562 496
1563 497
1564 498
1565 499
1566 500
1567 501
1568 502
1569 503
1570 504
1571 505
1572 506
1573 507
1574 508
1575 509
1576 510
1577 511
1578 512
1579 513
1580 514
1581 515
1582 516
1583 517
1584 518
1585 519
1586 520
1587 521
1588 522
1589 523
1590 524
1591 525
1592 526
1593 527
1594 528
1595 529
1596 530
1597 531
1598 532
1599 533
1600 534
1601 535
1602 536
1603 537
1604 538
1605 539
1606 540
1607 541
1608 542
1609 543
1610 544
1611 545
1612 546
1613 547
1614 548
1615 549
1616 550
1617 551
1618 552
1619 553
1620 554
1621 555
1622 556
1623 557
1624 558
1625 559
1626 560
1627 561
1628 562
1629 563
1630 564
1631 565
1632 566
1633 567
1634 568
1635 569
1636 570
1637 571
1638 572
1639 573
1640 574
1641 575
1642 576
1643 577
1644 578
1645 579
1646 580
1647 581
1648 582
1649 583
1650 584
1651 585
1652 586
1653 587
1654 588
1655 589
1656 590
1657 591
1658 592
1659 593
1660 594
1661 595
1662 596
1663 597
1664 598
1665 599
1666 600
1667 601
1668 602
1669 603
1670 604
1671 605
1672 606
1673 607
1674 608
1675 609
1676 610
1677 611
1678 612
1679 613
1680 614
1681 615
1682 616
1683 617
1684 618
1685 619
1686 620
1687 621
1688 622
1689 623
1690 624
1691 625
1692 626
1693 627
1694 628
1695 629
1696 630
1697 631
1698 632
1699 633
1700 634
1701 635
1702 636
1703 637
1704 638
1705 639
1706 640
1707 641
1708 642
1709 643
1710 644
1711 645
1712 646
1713 647
1714 648
1715 649
1716 650
1717 651
1718 652
1719 653
1720 654
1721 655
1722 656
1723 657
1724 658
1725 659
1726 660
1727 661
1728 662
1729 663
1730 664
1731 665
1732 666
1733 667
1734 668
1735 669
1736 670
1737 671
1738 672
1739 673
1740 674
1741 675
1742 676
1743 677
1744 678
1745 679
1746 680
1747 681
1748 682
1749 683
1750 684
1751 685
1752 686
1753 687
1754 688
1755 689
1756 690
1757 691
1758 692
1759 693
1760 694
1761 695
1762 696
1763 697
1764 698
1765 699
1766 700
1767 701
1768 702
1769 703
1770 704
1771 705
1772 706
1773 707
1774 708
1775 709
1776 710
1777 711
1778 712
1779 713
1780 714
1781 715
1782 716
1783 717
1784 718
1785 719
1786 720
1787 721
1788 722
1789 723
1790 724
1791 725
1792 726
1793 727
1794 728
1795 729
1796 730
1797 731
1798 732
1799 733
1800 734
1801 735
1802 736
1803 737
1804 738
1805 739
1806 740
1807 741
1808 742
1809 743
1810 744
1811 745
1812 746
1813 747
1814 748
1815 749
1816 750
1817 751
1818 752
1819 753
1820 754
1821 755
1822 756
1823 757
1824 758
1825 759
1826 760
1827 761
1828 762
1829 763
1830 764
1831 765
1832 766
1833 767
1834 768
1835 769
1836 770
1837 771
1838 772
1839 773
1840 774
1841 775
1842 776
1843 777
1844 778
1845 779
1846 780
1847 781
1848 782
1849 783
1850 784
1851 785
1852 786
1853 787
1854 788
1855 789
1856 790
1857 791
1858 792
1859 793
1860 794
1861 795
1862 796
1863 797
1864 798
1865 799
1866 800
1867 801
1868 802
1869 803
1870 804
1871 805
1872 806
1873 807
1874 808
1875 809
1876 810
1877 811
1878 812
1879 813
1880 814
1881 815
1882 816
1883 817
1884 818
1885 819
1886 820
1887 821
1888 822
1889 823
1890 824
1891 825
1892 826
1893 827
1894 828
1895 829
1896 830
1897 831
1898 832
1899 833
1900 834
1901 835
1902 836
1903 837
1904 838
1905 839
1906 840
1907 841
1908 842
1909 843
1910 844
1911 845
1912 846
1913 847
1914 848
1915 849
1916 850
1917 851
1918 852
1919 853
1920 854
1921 855
1922 856
1923 857
1924 858
1925 859
1926 860
1927 861
1928 862
1929 863
1930 864
1931 865
1932 866
1933 867
1934 868
1935 869
1936 870
1937 871
1938 872
1939 873
1940 874
1941 875
1942 876
1943 877
1944 878
1945 879
1946 880
1947 881
1948 882
1949 883
1950 884
1951 885
1952 886
1953 887
1954 888
1955 889
1956 890
1957 891
1958 892
1959 893
1960 894
1961 895
1962 896
1963 897
1964 898
1965 899
1966 900
1967 901
1968 902
1969 903
1970 904
1971 905
1972 906
1973 907
1974 908
1975 909
1976 910
1977 911
1978 912
1979 913
1980 914
1981 915
1982 916
1983 917
1984 918
1985 919
1986 920
1987 921
1988 922
1989 923
1990 924
1991 925
1992 926
1993 927
1994 928
1995 929
1996 930
1997 931
1998 932
1999 933
2000 934

```

Page 76 of 248	RSUStl_UnmarkObject	Fri Jan 04 16:35:25 2008	Page 76 of 248	RSUStl_UnmarkObject	Fri Jan 04 16:35:25 2008
1000 1001 1002 1003 1004 1005 1006 1007 1008 1009 1010 1011 1012 1013 1014 1015 1016 1017 1018 1019 1020 1021 1022 1023 1024 1025 1026 1027 1028 1029 1030 1031 1032 1033 1034 1035 1036 1037 1038 1039 1040 1041 1042 1043 1044 1045 1046 1047 1048 1049 1050 1051 1052 1053 1054 1055 1056 1057 1058 1059 1060 1061 1062 1063 1064 1065 1066 1067 1068 1069 1070 1071 1072 1073 1074 1075 1076 1077 1078 1079 1080 1081 1082 1083 1084 1085 1086 1087 1088 1089 1090 1091 1092 1093 1094 1095 1096 1097 1098 1099 1100 1101 1102 1103 1104 1105 1106 1107 1108 1109 1110 1111 1112 1113 1114 1115 1116 1117 1118 1119 1120 1121 1122 1123 1124 1125 1126 1127 1128 1129 1130 1131 1132 1133 1134 1135 1136 1137	<pre> /***** * UnmarkObject * * The unmarkObject operation takes a restorable object and unmarks * it, and * possibly its descendant files for restore based on the input * criteria. * The RSUStl_UnmarkObject call is an asynchronously executed * operation * in the Restore Engine that performs the unmarking. * It returns unmark * progress and tests for user cancelation through a callback * function. * * UnmarkObject Parameters: * * thisObject (I) - The restorable object; * can be a leaf object (e.g. a * file) or a container object (e.g., a directory). * * backupTime (I) - { * optional} The backup time to perform the unmark on -- * if not specified, * uses currently selected backup; if * specified, * leaves selected backup time unchanged * * descend (I) - allows unmark operation ONLY of files of state BADDATA. * * BadFileOnly (I) - * * BadFileCount (O) - returns the file count with BADDATA. * * FileMarked (O) - return the total files marked after this mark occurred. * * dirMarked (O) - return the total directories marked after this mark * occurred. * * otherMarked (I) - return the total "other" files marked after this mark. * * progress (I) - pointer to callback function to report progress and * test for cancelation *****/ extern_tlv RSUStl_UnmarkObject(struct RSRPC_User_RestorableObject *thisObject, const time_t backupTime, const boolean_t badFileOnly, const boolean_t badFileCount, const boolean_t fileMarked, const boolean_t otherMarked, const RSRPC_MarkProgressProc progressCB) { struct mark_context mc; extern_tlv rtc; time_t save_time = 0; u_hyper lenMarkedFiles; RSRPC_Backup_Flags_tlv backup_flags = 0; /* Initial global marking/unmarking cancel flag to PAUSE */ global_cancel_flag = FALSE; global_cancel_flag = TRUE; global_start_mark = rcp>rc_mark_total; } </pre>	<pre> 1139 1 mc.n_file = 0; 1140 1 mc.n_before = rcp>rc_marks_total; 1141 1 rcp>rc_cmd.Context = (char *)&mc; 1142 1 1143 1 1144 1 /* 1145 1 * Output parameters get initialized to zero 1146 1 */ 1147 1 1148 1 /* 1149 1 * Output parameters check pointers against NULL & initialized to 1150 1 zero 1151 1 */ 1152 1 1153 1 if (!fileMarked !dirMarked !otherMarked !badFileCount) 1154 2 { 1155 2 return EP_RA_RECOVER_BAD_ARGS; 1156 2 } 1157 2 1158 2 /* 1159 2 * Lets validate the restorable object as an input parameter. 1160 2 */ 1161 2 if (NULL == thisObject) 1162 2 { 1163 2 return EP_RA_RECOVER_BAD_ARGS; 1164 2 } 1165 2 1166 2 if (thisObject->root.objName == NULL) 1167 2 { 1168 2 return (EP_RA_RECOVER_INVALID_OBJNAME); 1169 2 } 1170 2 1171 2 switch (thisObject->root.objLevel) 1172 2 { 1173 2 case RSRPC_Container_Type: 1174 2 case RSRPC_Leaf_Type: 1175 2 /* make sure input object is from current TLO's backup app */ 1176 2 if (rcp>rc_backup_app != rcp->root.backupApp) 1177 2 { 1178 2 return (EP_RA_RECOVER_INVALID); 1179 2 } 1180 2 break; 1181 2 case RSRPC_tlv_Type: 1182 2 return (EP_RA_RECOVER_INVALID); 1183 2 } 1184 2 1185 2 default: 1186 2 return (EP_RA_RECOVER_INVALID_OBJTYPE); 1187 2 } 1188 2 1189 2 /* check if backup time must change */ 1190 2 if (backupTime != (time_t)0) 1191 2 { 1192 2 /* get current backup time, if different from requested time, 1193 2 * then set requested time and save current time; 1194 2 * current time */ 1195 2 rtc = rcp -> currentTlv-> pFuncArray[1196 2 pFuncIndexGetCurTime] 1197 2 { 1198 2 rcp, &save_time); 1199 2 if (rtc == RS_SUCCESS) </pre>			
Page 76 of 248	FSUnmarkum23	Fri Jan 04 16:35:25 2008	Page 76 of 248	FSUnmarkum24	Fri Jan 04 16:35:25 2008

```

1200 }      rtc = rcp -> currentPfcpt ->
1201         pFuncArray[
1202             PfuncIndexSelBackupPtime ]
1203     ) else {
1204         rtc = RSTSL_GetCurrentBackupPtime( save_time );
1205         if (rtc == E_SUCCESS)
1206             rtc = RSTSL_SetBackupPorTime (
1207                 backupTime, backup_flags);
1208     }
1209     if (rtc != E_SUCCESS)
1210         return rtc;
1211 }
1212 /* If this isn't a network backup restore,
1213 call plugin to do mark */
1214 if (rcp->rc_backup_app != 0)
1215 {
1216     rtc = rcp -> currentPfcpt-> pFuncArray[ PfuncIndexDmark ]
1217     {
1218         rcpt, thisObject, BadFilesOnly, descend,
1219         BadFilesCount, filemarked,
1220         klenMarkedFiles, dirmarked,
1221         othermarked,
1222         progress );
1223     }
1224 }
1225 /* save mark summary in restore context */
1226 rcp->rc_mark_summary.files = *filemarked;
1227 rcp->rc_mark_summary.dirs = *dirmarked;
1228 rcp->rc_mark_summary.other = *othermarked;
1229 rcp->rc_mark_summary.len_mkd_files.high = lenMarkedFiles_high;
1230 rcp->rc_mark_summary.len_mkd_files.low = lenMarkedFiles_low;
1231 if (save_time)
1232     rcp->currentPfcpt-> pFuncArray[
1233         PfuncIndexSelBackupPtime ]
1234     {
1235         rcpt, save_time, backup_flags);
1236     }
1237 return rtc;
1238 }
1239
1240 *filemarked = *dirmarked = *othermarked = *BadFilesCount = 0;
1241
1242 /*
1243 * The below is static variable global to this module
1244 ap1_unmarknum.c
1245 * They get initialized in EDWAS7_UnmarkObjecct to 0.
1246 * They get incremented in unmark_tree()
1247 * When the badfiles are encountered.
1248 * They get assigned to the output parameter to
1249   Edwos7_UnmarkObjecct after
1250   * the unmark_tree call.
1251 */
1252 BADMANF1UnmarkCount = 0;
1253
1254 /*
1255 * Input parameters
1256 * 1) BadFilesOnly -- is unmark limited to Badfiles only
1257 * 2) descend -- does unmark descend into the contents
1258   directories.
1259 */
1260 if (BadFilesOnly)

```

1284	2	{
1285	2	mc.no_badfiles = TRUE;
1286	1	}
1287	1	else
1288	2	{
1289	2	mc.no_badfiles = FALSE;
1290	1	}
1291	1	if (!descend)
1292	1	{
1293	1	mc.no_descend = TRUE;
1294	1	}
1295	1	else
1296	2	{
1297	2	mc.no_descend = FALSE;
1298	2	}
1299	1	}
1300	1	/*
1301	1	validate restorable object with the current mc context
1302	1	*/
1303	1	{
1304	1	tree_node *tmp_thisobj;
1305	1	int rc, umark_cree;
1306	1	tmp_thisobj = meat_lookup_path(rc->rc_mcp,
1307	1	tmp_thisobj->meat_lookup_path, thisobj->root.objName);
1308	1	{
1309	1	/*
1310	1	restorable object must be corrupt meat_lookup_path
1311	1	could not
1312	1	*/
1313	1	locate the tree node ptr for files
1314	1	{
1315	1	return EP_RB_RECOVER_BAD_CONTEXT;
1316	1	}
1317	1	if (tmp_thisobj->ri_mcpName !=
1318	1	(meat_lookup_path->ri_mcpName))
1319	1	{
1320	1	/*
1321	1	restorable object must be out of context return error
1322	1	*/
1323	1	return EP_RB_RECOVER_BAD_CONTEXT;
1324	1	}
1325	1	/*
1326	1	If we made it here then thisobj is valid.
1327	1	umark_cree always returns 0
1328	1	*/
1329	1	(void) umark_cree(tmp, tmp_thisobj);
1330	1	}
1331	1	/*
1332	1	Below are static variable global to this module
1333	1	api_markumark_c
1334	1	They get initialized in EMGST_umarkobj to 0.
1335	1	They get incremented in umark() when the badfiles are encountered.
1336	1	They get assigned to the output parameters for
1337	1	EMGST_umarkobj after
1338	1	The umark_cree call.
1339	1	*

Page 79 of 248	RSTSL_UnmarkObject	Fri Jan 04 16:35:25 2008	Page 80 of 248	MarkUnmarkDebugLogEcho	Fri Jan 04 16:35:25 2008
1315 1 1317 1 1319 1 1320 1 1321 1 1322 1 1323 1 1324 1 1325 1 1326 1 1327 1 1328 2 1329 2 1330 1 1331 1 1332 2 1333 2 1334 2 1335 2 1336 2 1338 2 1339 1 1341 1 1342 1 1344 1 1345 1	<pre>*/ *BadFilesCount=BADDATAFieldUnmarkCount; /* * If the summary is valid, set the total number of marked files, * directories, and "other" files. This is not intended to be the * number of files that resulted directly from the above * unmark_crow call. */ if (rcp->rc_mark_summary_valid) { *fileMarked = rcp->rc_mark_summary.nfiles; *dirMarked = rcp->rc_mark_summary.ndirs; *otherMarked = rcp->rc_mark_summary.nothers; } else { /* * This error is not fatal, the output variables *Marked, * *Will be zero, we should never get this error. */ rbe_log_status(0, "Internal error: mark summary not valid in RSTSL_UnmarkObject()"); } if (save_line) RSTSL_SetBackupForTime(save_line, backup_flags); return(E_SUCCESS); /* end of RSTSL_UnmarkObject () */</pre>	1347 1348 1349 1350 1351 1352 1353 1354 1356 2 1358 2 1359 1 1360 1 1361 1 1362 2 1363 2 1364 2 1367 2 1369 2 1370 1 1371 1 1372 1 1373 1	<pre>static void MarkUnmarkXDebugLogEcho(tree_node *tmp) { #if defined(DEBUG_LOG_MARK_UNMARK) { char pathbuf[EBL_MAXPATHLEN]; char *pnp = pathbuf; tn_getpath(tmp, &pnp); rbe_log_status(0, "The file %s was marked", pnp); } #endif #if defined(DEBUG) { char pathbuf[EBL_MAXPATHLEN]; char *pnp = pathbuf; tn_getpath(tmp, &pnp); printf("The file %s was marked\n", pnp); } #endif return; /* MarkUnmarkXDebugLogEcho */</pre>		

2	/*.....	53	/* Local headers
3	**	54	*/
4	** File Name: RSTSubmt.c	55	#include <RSLinterns.h>
5	**	56	#include <restore/EMWRESsubmtapi.h>
6	** Copyright (c) 1996,1999 by EMC Corporation.	57	
7	**	58	
8	** Purpose:	59	
9	**	60	
10	** The intent of the contents of this file is to implement the	61	
11	** functions to create the submobject and submtplment.	62	
12	**	63	
13	** These functions are provided to allow:	64	
14	- creation of submt objects, which define the set of objects to be	65	
15	restored and the scripts to be run before and after restoration,	66	
16	**	67	
17	** The following functions comprise Restoral management:	68	
18	**	69	
19	** RSTSL_Submt	70	
20	**	71	
21	** Compile-Time Options:	72	
22	** This section must list any compile time definitions	73	
23	** which will affect this header.	74	
24	**	75	
25	**	76	
26	*****	77	
27	*****	78	
28	*****	79	
29	*****	80	
30	*****	81	
31	*****	82	
32	*****	83	
33	*****	84	
34	*****	85	
35	*****	86	
36	*****	87	
37	*****	88	
38	*****	89	
39	*****	90	
40	*****	91	
41	*****	92	
42	*****	93	
43	*****	94	
44	*****	95	
45	*****	96	
46	*****	97	
47	*****	98	
48	*****	99	
49	*****	100	
50	*****	101	
51	*****	102	
52	*****	103	
53	*****	104	
54	*****	105	
55	*****	106	
56	*****	107	
57	*****	108	
58	*****	109	
59	*****	110	
60	*****	111	
61	*****	112	
62	*****	113	
63	*****	114	
64	*****	115	
65	*****	116	
66	*****	117	
67	*****	118	
68	*****	119	
69	*****	120	
70	*****	121	
71	*****	122	
72	*****	123	
73	*****	124	
74	*****	125	
75	*****	126	
76	*****	127	
77	*****	128	
78	*****	129	
79	*****	130	
80	*****	131	
81	*****	132	
82	*****	133	
83	*****	134	
84	*****	135	
85	*****	136	
86	*****	137	
87	*****	138	
88	*****	139	
89	*****	140	
90	*****	141	
91	*****	142	
92	*****	143	
93	*****	144	
94	*****	145	
95	*****	146	
96	*****	147	
97	*****	148	
98	*****	149	
99	*****	150	
100	*****	151	
101	*****	152	
102	*****	153	
103	*****	154	
104	*****	155	
105	*****	156	
106	*****	157	
107	*****	158	
108	*****	159	
109	*****	160	
110	*****	161	
111	*****	162	
112	*****	163	
113	*****	164	
114	*****	165	
115	*****	166	
116	*****	167	
117	*****	168	
118	*****	169	
119	*****	170	
120	*****	171	
121	*****	172	
122	*****	173	
123	*****	174	
124	*****	175	
125	*****	176	
126	*****	177	
127	*****	178	
128	*****	179	
129	*****	180	
130	*****	181	
131	*****	182	
132	*****	183	
133	*****	184	
134	*****	185	
135	*****	186	
136	*****	187	
137	*****	188	
138	*****	189	
139	*****	190	
140	*****	191	
141	*****	192	
142	*****	193	
143	*****	194	
144	*****	195	
145	*****	196	
146	*****	197	
147	*****	198	
148	*****	199	
149	*****	200	
150	*****	201	
151	*****	202	
152	*****	203	
153	*****	204	
154	*****	205	
155	*****	206	
156	*****	207	
157	*****	208	
158	*****	209	
159	*****	210	
160	*****	211	
161	*****	212	
162	*****	213	
163	*****	214	
164	*****	215	
165	*****	216	
166	*****	217	
167	*****	218	
168	*****	219	
169	*****	220	
170	*****	221	
171	*****	222	
172	*****	223	
173	*****	224	
174	*****	225	
175	*****	226	
176	*****	227	
177	*****	228	
178	*****	229	
179	*****	230	
180	*****	231	
181	*****	232	
182	*****	233	
183	*****	234	
184	*****	235	
185	*****	236	
186	*****	237	
187	*****	238	
188	*****	239	
189	*****	240	
190	*****	241	
191	*****	242	
192	*****	243	
193	*****	244	
194	*****	245	
195	*****	246	
196	*****	247	
197	*****	248	
198	*****	249	
199	*****	250	
200	*****	251	
201	*****	252	
202	*****	253	
203	*****	254	
204	*****	255	
205	*****	256	
206	*****	257	
207	*****	258	
208	*****	259	
209	*****	260	
210	*****	261	
211	*****	262	
212	*****	263	
213	*****	264	
214	*****	265	
215	*****	266	
216	*****	267	
217	*****	268	
218	*****	269	
219	*****	270	
220	*****	271	
221	*****	272	
222	*****	273	
223	*****	274	
224	*****	275	
225	*****	276	
226	*****	277	
227	*****	278	
228	*****	279	
229	*****	280	
230	*****	281	
231	*****	282	
232	*****	283	
233	*****	284	
234	*****	285	
235	*****	286	
236	*****	287	
237	*****	288	
238	*****	289	
239	*****	290	
240	*****	291	
241	*****	292	
242	*****	293	
243	*****	294	
244	*****	295	
245	*****	296	
246	*****	297	
247	*****	298	
248	*****	299	
249	*****	300	
250	*****	301	
251	*****	302	
252	*****	303	
253	*****	304	
254	*****	305	
255	*****	306	
256	*****	307	
257	*****	308	
258	*****	309	
259	*****	310	
260	*****	311	
261	*****	312	
262	*****	313	
263	*****	314	
264	*****	315	
265	*****	316	
266	*****	317	
267	*****	318	
268	*****	319	
269	*****	320	
270	*****	321	
271	*****	322	
272	*****	323	
273	*****	324	
274	*****	325	
275	*****	326	
276	*****	327	
277	*****	328	
278	*****	329	
279	*****	330	
280	*****	331	
281	*****	332	
282	*****	333	
283	*****	334	
284	*****	335	
285	*****	336	
286	*****	337	
287	*****	338	
288	*****	339	
289	*****	340	
290	*****	341	
291	*****	342	
292	*****	343	
293	*****	344	
294	*****	345	
295	*****	346	
296	*****	347	
297	*****	348	
298	*****	349	
299	*****	350	
300	*****	351	
301	*****	352	
302	*****	353	
303	*****	354	
304	*****	355	
305	*****	356	
306	*****	357	
307	*****	358	
308	*****	359	
309	*****	360	
310	*****	361	
311	*****	362	
312	*****	363	
313	*****	364	
314	*****	365	
315	*****	366	
316	*****	367	
317	*****	368	
318	*****	369	
319	*****	370	
320	*****	371	
321	*****	372	
322	*****	373	
323	*****	374	
324	*****	375	
325	*****	376	
326	*****	377	
327	*****	378	
328	*****	379	
329	*****	380	
330	*****	381	
331	*****	382	
332	*****	383	
333	*****	384	
334	*****	385	
335	*****	386	
336	*****	387	
337	*****	388	
338	*****	389	
339	*****	390	
340	*****	391	
341	*****	392	
342	*****	393	
343	*****	394	
344	*****	395	
345	*****	396	
346	*****	397	
347	*****	398	
348	*****	399	
349	*****	400	
350	*****	401	
351	*****	402	
352	*****	403	
353	*****	404	
354	*****	405	
355	*****	406	
356	*****	407	
357	*****	408	
358	*****	409	
359	*****	410	
360	*****	411	
361	*****	412	
362	*****	413	
363	*****	414	
364	*****	415	
365	*****	416	
366	*****	417	
367	*****	418	
368	*****	419	
369	*****	420	
370	*****	421	
371	*****	422	
372	*****	423	
373	*****	424	
374	*****	425	
375	*****	426	
376	*****	427	
377	*****	428	
378	*****	429	
379	*****	430	
380	*****	431	
381	*****	432	
382	*****	433	
383	*****		

```

128 *****
129 *
130 * Submit
131 *
132 * This function creates a submit object from the currently marked
133 * restorable objects. It is passed to RSTL_start to begin execution
134 * of the restore.
135 *
136 * Parameters:
137 *
138 * policy (I) - The overwrite policy to use
139 * inplace (I) - flag if the restore is to be in original locations
140 * hostname (I) - host to restore to (only if Inplace == False)
141 * directory (I) - directory to restore to
142 *
143 * transport (I) - Indicator of transport the restore is to be over (SCSI
144 * or network)
145 * submittobjID(I)
146 *
147 * ID - ID of the submit user object created to describe
148 * the number of total file objects submitted.
149 *
150 * progress (I) - pointer to callback function to report progress and
151 * test for cancellation
152 *
153 *
154 *
155 *
156 *
157 *
158 *
159 *
160 *
161 *
162 *
163 *
164 *
165 *
166 *
167 *
168 *
169 *
170 *
171 *
172 *
173 *
174 *
175 *
176 *
177 *
178 *
179 *
180 *
181 *
182 *
183 *
184 *
185 *
186 *
187 *
188 *
189 *
190 *
191 *
192 *
193 *
194 *
195 *
196 *
197 *
198 *
199 *
200 *
201 *
202 *
203 *
204 *
205 *
206 *
207 *
208 *
209 *
210 *
211 *
212 *
213 *
214 *
215 *
216 *
217 *
218 *
219 *
220 *
221 *
222 *
223 *
224 *
225 *
226 *
227 *
228 *
229 *
230 *
231 *
232 *
233 *
234 *
235 *
236 *
237 *
238 *
239 *
240 *
241 *
242 *
243 *
244 *
245 *
246 *
247 *
248 *
249 *
250 *
251 *
252 *
253 *
254 *
255 *
256 *
257 *
258 *
259 *
260 *
261 *
262 *
263 *
264 *
265 *
266 *
267 *
268 *
269 *
270 *
271 *
272 *
273 *
274 *
275 *
276 *
277 *
278 *
279 *
280 *
281 *
282 *
283 *
284 *
285 *
286 *
287 *
288 *
289 *
290 *
291 *
292 *
293 *
294 *
295 *
296 *
297 *
298 *
299 *
300 *
301 *
302 *
303 *
304 *
305 *
306 *
307 *
308 *
309 *
310 *
311 *
312 *
313 *
314 *
315 *
316 *
317 *
318 *
319 *
320 *
321 *
322 *
323 *
324 *
325 *
326 *
327 *
328 *
329 *
330 *
331 *
332 *
333 *
334 *
335 *
336 *
337 *
338 *
339 *
340 *
341 *
342 *
343 *
344 *
345 *
346 *
347 *
348 *
349 *
350 *
351 *
352 *
353 *
354 *
355 *
356 *
357 *
358 *
359 *
360 *
361 *
362 *
363 *
364 *
365 *
366 *
367 *
368 *
369 *
370 *
371 *
372 *
373 *
374 *
375 *
376 *
377 *
378 *
379 *
380 *
381 *
382 *
383 *
384 *
385 *
386 *
387 *
388 *
389 *
390 *
391 *
392 *
393 *
394 *
395 *
396 *
397 *
398 *
399 *
400 *
401 *
402 *
403 *
404 *
405 *
406 *
407 *
408 *
409 *
410 *
411 *
412 *
413 *
414 *
415 *
416 *
417 *
418 *
419 *
420 *
421 *
422 *
423 *
424 *
425 *
426 *
427 *
428 *
429 *
430 *
431 *
432 *
433 *
434 *
435 *
436 *
437 *
438 *
439 *
440 *
441 *
442 *
443 *
444 *
445 *
446 *
447 *
448 *
449 *
450 *
451 *
452 *
453 *
454 *
455 *
456 *
457 *
458 *
459 *
460 *
461 *
462 *
463 *
464 *
465 *
466 *
467 *
468 *
469 *
470 *
471 *
472 *
473 *
474 *
475 *
476 *
477 *
478 *
479 *
480 *
481 *
482 *
483 *
484 *
485 *
486 *
487 *
488 *
489 *
490 *
491 *
492 *
493 *
494 *
495 *
496 *
497 *
498 *
499 *
500 *
501 *
502 *
503 *
504 *
505 *
506 *
507 *
508 *
509 *
510 *
511 *
512 *
513 *
514 *
515 *
516 *
517 *
518 *
519 *
520 *
521 *
522 *
523 *
524 *
525 *
526 *
527 *
528 *
529 *
530 *
531 *
532 *
533 *
534 *
535 *
536 *
537 *
538 *
539 *
540 *
541 *
542 *
543 *
544 *
545 *
546 *
547 *
548 *
549 *
550 *
551 *
552 *
553 *
554 *
555 *
556 *
557 *
558 *
559 *
560 *
561 *
562 *
563 *
564 *
565 *
566 *
567 *
568 *
569 *
570 *
571 *
572 *
573 *
574 *
575 *
576 *
577 *
578 *
579 *
580 *
581 *
582 *
583 *
584 *
585 *
586 *
587 *
588 *
589 *
590 *
591 *
592 *
593 *
594 *
595 *
596 *
597 *
598 *
599 *
600 *
601 *
602 *
603 *
604 *
605 *
606 *
607 *
608 *
609 *
610 *
611 *
612 *
613 *
614 *
615 *
616 *
617 *
618 *
619 *
620 *
621 *
622 *
623 *
624 *
625 *
626 *
627 *
628 *
629 *
630 *
631 *
632 *
633 *
634 *
635 *
636 *
637 *
638 *
639 *
640 *
641 *
642 *
643 *
644 *
645 *
646 *
647 *
648 *
649 *
650 *
651 *
652 *
653 *
654 *
655 *
656 *
657 *
658 *
659 *
660 *
661 *
662 *
663 *
664 *
665 *
666 *
667 *
668 *
669 *
670 *
671 *
672 *
673 *
674 *
675 *
676 *
677 *
678 *
679 *
680 *
681 *
682 *
683 *
684 *
685 *
686 *
687 *
688 *
689 *
690 *
691 *
692 *
693 *
694 *
695 *
696 *
697 *
698 *
699 *
700 *
701 *
702 *
703 *
704 *
705 *
706 *
707 *
708 *
709 *
710 *
711 *
712 *
713 *
714 *
715 *
716 *
717 *
718 *
719 *
720 *
721 *
722 *
723 *
724 *
725 *
726 *
727 *
728 *
729 *
730 *
731 *
732 *
733 *
734 *
735 *
736 *
737 *
738 *
739 *
740 *
741 *
742 *
743 *
744 *
745 *
746 *
747 *
748 *
749 *
750 *
751 *
752 *
753 *
754 *
755 *
756 *
757 *
758 *
759 *
760 *
761 *
762 *
763 *
764 *
765 *
766 *
767 *
768 *
769 *
770 *
771 *
772 *
773 *
774 *
775 *
776 *
777 *
778 *
779 *
780 *
781 *
782 *
783 *
784 *
785 *
786 *
787 *
788 *
789 *
790 *
791 *
792 *
793 *
794 *
795 *
796 *
797 *
798 *
799 *
800 *
801 *
802 *
803 *
804 *
805 *
806 *
807 *
808 *
809 *
810 *
811 *
812 *
813 *
814 *
815 *
816 *
817 *
818 *
819 *
820 *
821 *
822 *
823 *
824 *
825 *
826 *
827 *
828 *
829 *
830 *
831 *
832 *
833 *
834 *
835 *
836 *
837 *
838 *
839 *
840 *
841 *
842 *
843 *
844 *
845 *
846 *
847 *
848 *
849 *
850 *
851 *
852 *
853 *
854 *
855 *
856 *
857 *
858 *
859 *
860 *
861 *
862 *
863 *
864 *
865 *
866 *
867 *
868 *
869 *
870 *
871 *
872 *
873 *
874 *
875 *
876 *
877 *
878 *
879 *
880 *
881 *
882 *
883 *
884 *
885 *
886 *
887 *
888 *
889 *
890 *
891 *
892 *
893 *
894 *
895 *
896 *
897 *
898 *
899 *
900 *
901 *
902 *
903 *
904 *
905 *
906 *
907 *
908 *
909 *
910 *
911 *
912 *
913 *
914 *
915 *
916 *
917 *
```

```

189 1
190 1
191 1
192 1
193 1
194 1
195 1
196 1
197 1
198 1
199 1
200 1
201 1
202 1
203 1
204 1
205 1
206 1
207 1
208 1
209 1
210 1
211 1
212 1
213 1
214 1
215 1
216 1
217 1
218 1
219 1
220 1
221 1
222 1
223 1
224 1
225 1
226 1
227 1
228 1
229 1
230 1
231 1
232 1
233 1
234 1
235 1
236 1
237 1
238 1
239 1
240 1
241 1
242 1
243 1
244 1
245 1
246 1
247 1
248 1
249 1
250 1
251 1
252 1
253 1
254 1
255 1
256 1
257 1
258 1
259 1
260 1
261 1
262 1
263 1
264 1
265 1
266 1
267 1
268 1
269 1
270 1
271 1
272 1
273 1
274 1
275 1
276 1
277 1
278 1
279 1
280 1
281 1
282 1
283 1
284 1
285 1
286 1
287 1
288 1
289 1
290 1
291 1
292 1
293 1
294 1
295 1
296 1
297 1
298 1
299 1
300 1
301 1
302 1
303 1
304 1
305 1
306 1
307 1
308 1
309 1
310 1
311 1
312 1
313 1
314 1
315 1
316 1
317 1
318 1
319 1
320 1
321 1
322 1
323 1
324 1
325 1
326 1
327 1
328 1
329 1
330 1
331 1
332 1
333 1
334 1
335 1
336 1
337 1
338 1
339 1
340 1
341 1
342 1
343 1
344 1
345 1
346 1
347 1
348 1
349 1
350 1
351 1
352 1
353 1
354 1
355 1
356 1
357 1
358 1
359 1
360 1
361 1
362 1
363 1
364 1
365 1
366 1
367 1
368 1
369 1
370 1
371 1
372 1
373 1
374 1
375 1
376 1
377 1
378 1
379 1
380 1
381 1
382 1
383 1
384 1
385 1
386 1
387 1
388 1
389 1
390 1
391 1
392 1
393 1
394 1
395 1
396 1
397 1
398 1
399 1
400 1
401 1
402 1
403 1
404 1
405 1
406 1
407 1
408 1
409 1
410 1
411 1
412 1
413 1
414 1
415 1
416 1
417 1
418 1
419 1
420 1
421 1
422 1
423 1
424 1
425 1
426 1
427 1
428 1
429 1
430 1
431 1
432 1
433 1
434 1
435 1
436 1
437 1
438 1
439 1
440 1
441 1
442 1
443 1
444 1
445 1
446 1
447 1
448 1
449 1
450 1
451 1
452 1
453 1
454 1
455 1
456 1
457 1
458 1
459 1
460 1
461 1
462 1
463 1
464 1
465 1
466 1
467 1
468 1
469 1
470 1
471 1
472 1
473 1
474 1
475 1
476 1
477 1
478 1
479 1
480 1
481 1
482 1
483 1
484 1
485 1
486 1
487 1
488 1
489 1
490 1
491 1
492 1
493 1
494 1
495 1
496 1
497 1
498 1
499 1
500 1
501 1
502 1
503 1
504 1
505 1
506 1
507 1
508 1
509 1
510 1
511 1
512 1
513 1
514 1
515 1
516 1
517 1
518 1
519 1
520 1
521 1
522 1
523 1
524 1
525 1
526 1
527 1
528 1
529 1
530 1
531 1
532 1
533 1
534 1
535 1
536 1
537 1
538 1
539 1
540 1
541 1
542 1
543 1
544 1
545 1
546 1
547 1
548 1
549 1
550 1
551 1
552 1
553 1
554 1
555 1
556 1
557 1
558 1
559 1
560 1
561 1
562 1
563 1
564 1
565 1
566 1
567 1
568 1
569 1
570 1
571 1
572 1
573 1
574 1
575 1
576 1
577 1
578 1
579 1
580 1
581 1
582 1
583 1
584 1
585 1
586 1
587 1
588 1
589 1
590 1
591 1
592 1
593 1
594 1
595 1
596 1
597 1
598 1
599 1
600 1
601 1
602 1
603 1
604 1
605 1
606 1
607 1
608 1
609 1
610 1
611 1
612 1
613 1
614 1
615 1
616 1
617 1
618 1
619 1
620 1
621 1
622 1
623 1
624 1
625 1
626 1
627 1
628 1
629 1
630 1
631 1
632 1
633 1
634 1
635 1
636 1
637 1
638 1
639 1
640 1
641 1
642 1
643 1
644 1
645 1
646 1
647 1
648 1
649 1
650 1
651 1
652 1
653 1
654 1
655 1
656 1
657 1
658 1
659 1
660 1
661 1
662 1
663 1
664 1
665 1
666 1
667 1
668 1
669 1
670 1
671 1
672 1
673 1
674 1
675 1
676 1
677 1
678 1
679 1
680 1
681 1
682 1
683 1
684 1
685 1
686 1
687 1
688 1
689 1
690 1
691 1
692 1
693 1
694 1
695 1
696 1
697 1
698 1
699 1
700 1
701 1
702 1
703 1
704 1
705 1
706 1
707 1
708 1
709 1
710 1
711 1
712 1
713 1
714 1
715 1
716 1
717 1
718 1
719 1
720 1
721 1
722 1
723 1
724 1
725 1
726 1
727 1
728 1
729 1
730 1
731 1
732 1
733 1
734 1
735 1
736 1
737 1
738 1
739 1
740 1
741 1
742 1
743 1
744 1
745 1
746 1
747 1
748 1
749 1
750 1
751 1
752 1
753 1
754 1
755 1
756 1
757 1
758 1
759 1
760 1
761 1
762 1
763 1
764 1
765 1
766 1
767 1
768 1
769 1
770 1
771 1
772 1
773 1
774 1
775 1
776 1
777 1
778 1
779 1
780 1
781 1
782 1
783 1
784 1
785 1
786 1
787 1
788 1
789 1
790 1
791 1
792 1
793 1
794 1
795 1
796 1
797 1
798 1
799 1
800 1
801 1
802 1
803 1
804 1
805 1
806 1
807 1
808 1
809 1
810 1
811 1
812 1
813 1
814 1
815 1
816 1
817 1
818 1
819 1
820 1
821 1
822 1
823 1
824 1
825 1
826 1
827 1
828 1
829 1
830 1
831 1
832 1
833 1
834 1
835 1
836 1
837 1
838 1
839 1
840 1
841 1
842 1
843 1
844 1
845 1
846 1
847 1
848 1
849 1
850 1
851 1
852 1
853 1
854 1
855 1
856 1
857 1
858 1
859 1
860 1
861 1
862 1
863 1
864 1
865 1
866 1
867 1
868 1
869 1
870 1
8
```



```

Page 67 of 248
RSTSL_Submit
Fri Jan 04 16:35:25 2008

317 1         if (!inPlace)
318 2         {
319 3             if (NULL == rcp->rc_source_client_hostname)
320 4             {
321 5                 return (EP_RR_RECOVER_BAD_CONTEXT);
322 6             }
323 7             else
324 8             {
325 9                 hostname_SE = ssl_strdup(rcp->rc_source_client_hostname);
326 10            }
327 11        }
328 12        }
329 13        }
330 14        }
331 15        }
332 16        }
333 17        }
334 18        }
335 19        }
336 20        }
337 21        }
338 22        }
339 23        }
340 24        }
341 25        }
342 26        }
343 27        }
344 28        }
345 29        }
346 30        }
347 31        }
348 32        }
349 33        }
350 34        }
351 35        }
352 36        }
353 37        }
354 38        }
355 39        }
356 40        }
357 41        }
358 42        }
359 43        }
360 44        }
361 45        }
362 46        }
363 47        }
364 48        }
365 49        }
366 50        }
367 51        }
368 52        }
369 53        }
370 54        }
371 55        }
372 56        }
373 57        }
374 58        }
375 59        }
376 60        }
377 61        }
378 62        }
379 63        }
380 64        }
381 65        }
382 66        }
383 67        }
384 68        }
385 69        }
386 70        }
387 71        }
388 72        }
389 73        }
390 74        }
391 75        }
392 76        }
393 77        }
394 78        }
395 79        }
396 80        }
397 81        }
398 82        }
399 83        }
400 84        }
401 85        }
402 86        }
403 87        }
404 88        }
405 89        }
406 90        }
407 91        }
408 92        }
409 93        }
410 94        }
411 95        }
412 96        }
413 97        }
414 98        }
415 99        }
416 100       }
417 101       }
418 102       }
419 103       }
420 104       }
421 105       }
422 106       }
423 107       }
424 108       }
425 109       }
426 110       }
427 111       }
428 112       }
429 113       }
430 114       }
431 115       }
432 116       }
433 117       }
434 118       }
435 119       }
436 120       }
437 121       }
438 122       }
439 123       }
440 124       }
441 125       }
442 126       }
443 127       }
444 128       }
445 129       }
446 130       }
447 131       }
448 132       }
449 133       }
450 134       }
451 135       }
452 136       }
453 137       }
454 138       }
455 139       }
456 140       }
457 141       }
458 142       }
459 143       }
460 144       }
461 145       }
462 146       }
463 147       }
464 148       }
465 149       }
466 150       }
467 151       }
468 152       }
469 153       }
470 154       }
471 155       }
472 156       }
473 157       }
474 158       }
475 159       }
476 160       }
477 161       }
478 162       }
479 163       }
480 164       }
481 165       }
482 166       }
483 167       }
484 168       }
485 169       }
486 170       }
487 171       }
488 172       }
489 173       }
490 174       }
491 175       }
492 176       }
493 177       }
494 178       }
495 179       }
496 180       }
497 181       }
498 182       }
499 183       }
500 184       }
501 185       }
502 186       }
503 187       }
504 188       }
505 189       }
506 190       }
507 191       }
508 192       }
509 193       }
510 194       }
511 195       }
512 196       }
513 197       }
514 198       }
515 199       }
516 200       }
517 201       }
518 202       }
519 203       }
520 204       }
521 205       }
522 206       }
523 207       }
524 208       }
525 209       }
526 210       }
527 211       }
528 212       }
529 213       }
530 214       }
531 215       }
532 216       }
533 217       }
534 218       }
535 219       }
536 220       }
537 221       }
538 222       }
539 223       }
540 224       }
541 225       }
542 226       }
543 227       }
544 228       }
545 229       }
546 230       }
547 231       }
548 232       }
549 233       }
550 234       }
551 235       }
552 236       }
553 237       }
554 238       }
555 239       }
556 240       }
557 241       }
558 242       }
559 243       }
560 244       }
561 245       }
562 246       }
563 247       }
564 248       }
565 249       }
566 250       }
567 251       }
568 252       }
569 253       }
570 254       }
571 255       }
572 256       }
573 257       }
574 258       }
575 259       }
576 260       }
577 261       }
578 262       }
579 263       }
580 264       }
581 265       }
582 266       }
583 267       }
584 268       }
585 269       }
586 270       }
587 271       }
588 272       }
589 273       }
590 274       }
591 275       }
592 276       }
593 277       }
594 278       }
595 279       }
596 280       }
597 281       }
598 282       }
599 283       }
600 284       }
601 285       }
602 286       }
603 287       }
604 288       }
605 289       }
606 290       }
607 291       }
608 292       }
609 293       }
610 294       }
611 295       }
612 296       }
613 297       }
614 298       }
615 299       }
616 300       }
617 301       }
618 302       }
619 303       }
620 304       }
621 305       }
622 306       }
623 307       }
624 308       }
625 309       }
626 310       }
627 311       }
628 312       }
629 313       }
630 314       }
631 315       }
632 316       }
633 317       }
634 318       }
635 319       }
636 320       }
637 321       }
638 322       }
639 323       }
640 324       }
641 325       }
642 326       }
643 327       }
644 328       }
645 329       }
646 330       }
647 331       }
648 332       }
649 333       }
650 334       }
651 335       }
652 336       }
653 337       }
654 338       }
655 339       }
656 340       }
657 341       }
658 342       }
659 343       }
660 344       }
661 345       }
662 346       }
663 347       }
664 348       }
665 349       }
666 350       }
667 351       }
668 352       }
669 353       }
670 354       }
671 355       }
672 356       }
673 357       }
674 358       }
675 359       }
676 360       }
677 361       }
678 362       }
679 363       }
680 364       }
681 365       }
682 366       }
683 367       }
684 368       }
685 369       }
686 370       }
687 371       }
688 372       }
689 373       }
690 374       }
691 375       }
692 376       }
693 377       }
694 378       }
695 379       }
696 380       }
697 381       }
698 382       }
699 383       }
700 384       }
701 385       }
702 386       }
703 387       }
704 388       }
705 389       }
706 390       }
707 391       }
708 392       }
709 393       }
710 394       }
711 395       }
712 396       }
713 397       }
714 398       }
715 399       }
716 400       }
717 401       }
718 402       }
719 403       }
720 404       }
721 405       }
722 406       }
723 407       }
724 408       }
725 409       }
726 410      
```

```

440 1         {
441 2             return (EP_RB_RECOVER_FATALERR);
442 3         }
443 4     }
444 5     else
445 6     {
446 7         rbe_log_state(
447 8             0, "Restore context has not set template_name,
448 9             witem_name and/or source_client_name, in RSTSL.Submit()");
449 10        return(EP_RB_RECOVER_BAD_CONTEXT);
450 11    }
451 12
452 13    /* directory & hostname are check as input args */
453 14    /* What about transport ?? */
454 15
455 16    if(0 != SetSeedInitiation(&submitObjID,
456 17                             submitLEIDID,
457 18                             (char *)hostname_SB,
458 19                             inPlace,
459 20                             (char *)directory,
460 21                             policy,
461 22                             inPlace,
462 23                             &SeedStatus))
463 24    {
464 25        return (EP_RB_RECOVER_FATALERR);
465 26    }
466 27
467 28    if(NULL != rcp->rc_client_dttrop)
468 29    {
469 30        if(0 != SetSeedInitop(&submitObjID,
470 31                              submitLEIDID,
471 32                              rcp->rc_client_dttrop,
472 33                              rcp->rc_client_dttrop,
473 34                              &SeedStatus))
474 35        {
475 36            return (EP_RB_RECOVER_FATALERR);
476 37        }
477 38    }
478 39    else
479 40    {
480 41        rbe_log_state(
481 42            0, "Restore context has not set dttrop, in RSTSL.Submit()");
482 43        return(EP_RB_RECOVER_BAD_CONTEXT);
483 44    }
484 45
485 46    if(NULL != rcp->rc_client_dttrop)
487 47    {
488 48        if(0 != SetSeedScriptName(&submitObjID,
489 49                                  submitLEIDID,
490 50                                  rcp->rc_client_scriptname,
491 51                                  rcp->rc_client_dttropname,
492 52                                  &SeedStatus))
493 53        {
494 54            return (EP_RB_RECOVER_FATALERR);
495 55        }
496 56    }
497 57
498 58    return (0);
499 59
500 60    }
501 61
502 62    }
503 63
504 64    }
505 65
506 66    }
507 67
508 68    }
509 69
510 70    }
511 71
512 72    }
513 73
514 74    }
515 75
516 76    }
517 77
518 78    }
519 79
520 80    }
521 81
522 82    }
523 83
524 84    }
525 85
526 86    }
527 87
528 88    }
529 89
530 90    }
531 91
532 92    }
533 93
534 94    }
535 95
536 96    }
537 97
538 98    }
539 99
540 100   }
541 101
542 102   }
543 103
544 104   }
545 105
546 106   }
547 107
548 108   }
549 109
550 110   }
551 111
552 112   }
553 113
554 114   }
555 115
556 116   }
557 117
558 118   }
559 119
560 120   }
561 121
562 122   }
563 123
564 124   }
565 125
566 126   }
567 127
568 128   }
569 129
570 130   }
571 131
572 132   }
573 133
574 134   }
575 135
576 136   }
577 137
578 138   }
579 139
580 140   }
581 141
582 142   }
583 143
584 144   }
585 145
586 146   }
587 147
588 148   }
589 149
590 150   }
591 151
592 152   }
593 153
594 154   }
595 155
596 156   }
597 157
598 158   }
599 159
600 160   }
601 161
602 162   }
603 163
604 164   }
605 165
606 166   }
607 167
608 168   }
609 169
610 170   }
611 171
612 172   }
613 173
614 174   }
615 175
616 176   }
617 177
618 178   }
619 179
620 180   }
621 181
622 182   }
623 183
624 184   }
625 185
626 186   }
627 187
628 188   }
629 189
630 190   }
631 191
632 192   }
633 193
634 194   }
635 195
636 196   }
637 197
638 198   }
639 199
640 200   }
641 201
642 202   }
643 203
644 204   }
645 205
646 206   }
647 207
648 208   }
649 209
650 210   }
651 211
652 212   }
653 213
654 214   }
655 215
656 216   }
657 217
658 218   }
659 219
660 220   }
661 221
662 222   }
663 223
664 224   }
665 225
666 226   }
667 227
668 228   }
669 229
670 230   }
671 231
672 232   }
673 233
674 234   }
675 235
676 236   }
677 237
678 238   }
679 239
680 240   }
681 241
682 242   }
683 243
684 244   }
685 245
686 246   }
687 247
688 248   }
689 249
690 250   }
691 251
692 252   }
693 253
694 254   }
695 255
696 256   }
697 257
698 258   }
699 259
700 260   }
701 261
702 262   }
703 263
704 264   }
705 265
706 266   }
707 267
708 268   }
709 269
710 270   }
711 271
712 272   }
713 273
714 274   }
715 275
716 276   }
717 277
718 278   }
719 279
720 280   }
721 281
722 282   }
723 283
724 284   }
725 285
726 286   }
727 287
728 288   }
729 289
730 290   }
731 291
732 292   }
733 293
734 294   }
735 295
736 296   }
737 297
738 298   }
739 299
740 300   }
741 301
742 302   }
743 303
744 304   }
745 305
746 306   }
747 307
748 308   }
749 309
750 310   }
751 311
752 312   }
753 313
754 314   }
755 315
756 316   }
757 317
758 318   }
759 319
760 320   }
761 321
762 322   }
763 323
764 324   }
765 325
766 326   }
767 327
768 328   }
769 329
770 330   }
771 331
772 332   }
773 333
774 334   }
775 335
776 336   }
777 337
778 338   }
779 339
780 340   }
781 341
782 342   }
783 343
784 344   }
785 345
786 346   }
787 347
788 348   }
789 349
790 350   }
791 351
792 352   }
793 353
794 354   }
795 355
796 356   }
797 357
798 358   }
799 359
800 360   }
801 361
802 362   }
803 363
804 364   }
805 365
806 366   }
807 367
808 368   }
809 369
810 370   }
811 371
812 372   }
813 373
814 374   }
815 375
816 376   }
817 377
818 378   }
819 379
820 380   }
821 381
822 382   }
823 383
824 384   }
825 385
826 386   }
827 387
828 388   }
829 389
830 390   }
831 391
832 392   }
833 393
834 394   }
835 395
836 396   }
837 397
838 398   }
839 399
840 400   }
841 401
842 402   }
843 403
844 404   }
845 405
846 406   }
847 407
848 408   }
849 409
850 410   }
851 411
852 412   }
853 413
854 414   }
855 415
856 416   }
857 417
858 418   }
859 419
860 420   }
861 421
862 422   }
863 423
864 424   }
865 425
866 426   }
867 427
868 428   }
869 4
```

```

497 2 {
498 2     the_log_state(
499 2         0, "Restore context has not set scriptname or client user name,
500 1         return(EP_RB_RECOVER_BAD_CONTEXT);
501 1         In RSTSL_Submit()");
502 1     }
503 1     /*
504 1     * Progress callback intended to test for cancellation and
505 1     * to report progress on the submit to the user.
506 1     */
507 2     if(TRUE == progressCB(0))
508 2     {
509 2         /* Clean up here!
510 2         * Right now there is no clean up routine for submitObjects.
511 2         */
512 2         the_log_state(EP_RB_RECOVER_ABORT, "User abort during submit.");
513 2         submitCancelled = TRUE;
514 2         return(EP_RB_RECOVER_ABORT);
515 1     }
516 1     submit_fd = OpenSubmitFile(TRUE,
517 1         /*submitObjID,
518 1         submitElemID,
519 1         sSubmitus);
520 1     }
521 1     if(!-1 == submit_fd)
522 2     {
523 2         return (EP_RB_RECOVER_FATALERR);
524 1     }
525 1     ret_status = push_submit_file(rcp,
526 1         submit_fd,
527 1         /*submitObjID,
528 1         submitElemID,
529 1         &this_submit_volumes,
530 1         local_submit_files,
531 1         local_submit_volumes,
532 1         progressCB,
533 1         submitCancelled);
534 1     }
535 1     CloseSubmitFile(submit_fd, TRUE, sSubmitus);
536 1     if(TRUE == submitCancelled)
537 2     {
538 2         /* Lets clean up here!
539 2         * Right now there is no clean up routine for submitObjects.
540 2         */
541 2         the_log_state(EP_RB_RECOVER_ABORT, "User abort during submit.");
542 2         return(EP_RB_RECOVER_ABORT);
543 1     }
544 1     if(-1 == ret_status)
545 2     {
546 2         return (EP_RB_RECOVER_FATALERR);
547 1     }
548 1     *ObjectsSubmitted = (unsigned int) ret_status;
549 1     if(0 != SetSocketSize(*submitObjID,
550 1         local_submit_files,
551 1         sSubmitus))
552 2     {
553 2         /* Not sure this one needs to be handled */
554 1     }
555 1 }

```

```

561 1     if(0 != SetSocketVolumes(*submitObjID,
562 1         this_submit_files, sSubmitus))
563 2     {
564 2         /* Not sure this one needs to be handled */
565 2     }
566 1     if(0 != SetSummary(*submitObjID, submitElemID,
567 1         this_submit_files, sSubmitus))
568 2     {
569 2         /* Not sure this one needs to be handled */
570 2     }
571 1     if(0 != SetVolumes(*submitObjID,
572 1         submitElemID,
573 1         this_submit_volumes,
574 1         sSubmitus))
575 2     {
576 2         /* Not sure this one needs to be handled */
577 2     }
578 1     return( E_SUCCESS );
579 1 }
580 1 }

```

```

586 void
587 fill_client_dirtop(struct recover_context *rcx)
588 {
589     char buf[4096];
590     RBC_WORKITEM *pwi;
591     RBC_WORKGROUP *pwg;
592     boolean_t cross_recover;
593
594     for (pwg = rcx->rc_config->pgrouplist; NULL != pwg;
595          pwg = pwg->next)
596     {
597         for (pwi = pwg->pwlist; NULL != pw; pw = pw->next)
598         {
599             if (0 == strcmp(pwi->name, rcx->rc_workitem_name))
600             {
601                 goto gottit;
602             }
603         }
604     }
605
606     gottit2:
607
608     /*
609      * If the dirtop already specifies a network client
610      * target, remove it first.
611      */
612
613     if (rc->rc_client_dirtop != NULL
614         && NULL != strchr(rcx->rc_client_dirtop, ':'))
615     {
616         return;
617     }
618
619     /*
620      * cross_recover is a boolean variable used to indicate a
621      * This will set the proper target for NOS clients and SHOULD NOT
622      * direct others.
623      */
624
625     cross_recover = (NULL != rcx->rc_client_hostname) &&
626                     (0 != strcmp(
627                         rcx->rc_source_client_hostname, rcx->rc_client_hostname));
628
629     sprintf(buf, "%s%s",
630            (NULL != pwg && NULL != pw->nw_cint_target)
631            ? (cross_recover
632              ? rcx->rc_client_hostname
633              : pw->nw_cint_target)
634            : ""),
635            ((NULL != pw) && NULL != pw->nw_cint_target) ? ":" : "");
636
637     NULL := rcx->rc_client_dirtop;
638     if (rcx->rc_client_dirtop != NULL)
639     {
640         free (rcx->rc_client_dirtop);
641     }
642
643     rcx->rc_client_dirtop = buf;
644     if (NULL == rcx->rc_client_dirtop)
645     {
646

```

```

646     rcx->rc_log_cant_find_CSX_WORKITEM, NULL));
647 }
648 /* fill_client_dirtop */

```

```

652 static int
653 push_submtc_file(struct recover_context *rcx,
654                  int submtc_id,
655                  struct mark_summary *this_submtc_files,
656                  struct mark_summary *this_submtc_volumes,
657                  struct mark_summary *total_submtc_files,
658                  struct mark_summary *total_submtc_volumes,
659                  RSFSI_SubmtcProgressProc progressCB,
660                  boolean_t *submtcCancelled)
661 {
662     int submtc_count = 1;
663     int retestStatus = 0;
664     int bitfiles_pushed = 0;
665     ebv_euid_t prev_euid;
666
667     if (NULL != submtcCancelled)
668         *submtcCancelled = FALSE;
669     else
670         return -1;
671
672     memset(&prev_euid, 0, sizeof(ebv_euid_t));

```

```

673     if ((NULL == rcx) ||
674         (NULL == this_submtc_files) ||
675         (NULL == this_submtc_volumes) ||
676         (NULL == total_submtc_files) ||
677         (NULL == total_submtc_volumes))
678     {
679         return -1;
680     }
681     if (submtc_count)
682     {
683         inc plane;
684         /* MCAT TOP is defining in mcat.h */
685         for (plane = (-rcx->rc_planes)+1; plane <= MCAT_TOP;
686             plane++)

```

```

687     {
688         cat_descriptor *card = mcat_getcard(rcx->rc_mcp, plane);
689         char *marks = rcx->rc_marks[plane];
690         long limo;
691         if (card == NULL) /* should never happen */
692         {
693             continue;
694         }
695         /* carddesc.h */
696         nrlim = card_nrlim(card);
697         for (limo = 0; limo < nrlim; limo++)
698         {
699             /* RSISubmtcIn.h */
700             if (! TLIMNO_MARKED(marks, limo))
701             {
702                 continue;
703             }

```

```

704         {
705             continue;
706         }
707         /* RSISubmtcIn.h */
708         if (! TLIMNO_MARKED(marks, limo))
709         {
710             continue;
711         }
712     }

```

```

713     retestStatus = push_binfo_to_submtcfile(rcx,
714                                             plane, card,
715                                             submtc_id,
716                                             &prev_euid,
717                                             this_submtc_files,
718                                             this_submtc_volumes,
719                                             total_submtc_files,
720                                             total_submtc_volumes);
721     if (1 == retestStatus)
722     {
723         bitfiles_pushed++;
724     }
725     if (bitfiles_pushed & 1024)
726     {
727         if (TRUE == progressCB(bitfiles_pushed))
728         {
729             /* Log status (E.D. RECOVER_ABORT,
730              * "submtcCancelled" = TRUE,
731              * return(-1)). */
732             return(-1);
733         }
734     }
735     if (1 == retestStatus)
736     {
737         if (-1 == retestStatus)
738         {
739             return -1;
740         }
741         return bitfiles_pushed;
742     }
743     }
744 }

```



```

879 1      /* If this is a renamed element,
880 1      /*
881 1      must get the current name from cat
882 1
883 1      (void)card_read_cat(catin,card, catlno, kclm, kfname);
884 1      if (cjm.ce_status & CESTAT_RENAME)
885 2      {
886 2          /*
887 2          *name may contain subscripts as in newware
888 2          */
889 2          namesize = (size_t)cjm.ce.namesize;
890 2      }
891 1      if(f0 = getDobid(kclm.ce.swid, kselid))
892 1      {
893 1          /*_log_stats(
894 1          0, *** warning: could not determine bitfile directory for "
895 2          "file \"%s\", skipping it.", fname);
896 2          return 0;
897 2      }
898 1      if ((NULL != rcx->rcex_directives_D) || (NULL != fname))
899 1      {
900 1          directives_list=BSTL_get_directives_for_file(
901 2          rcx->rcex_directives_D,
902 2          fname);
903 2      }
904 1      else
905 1      {
906 1          directives_list = NULL;
907 2      }
908 1      /* done in case string not null terminated */
909 1      if(directives_list != NULL)
910 1      {
911 1          directives_size = strlen(directives_list);
912 2      }
913 1      else
914 1      {
915 2          directives_size = 0;
916 2      }
917 1      if(f0 = writeBifileInfoFromSubmFile(f0,
918 2          kselid,
919 2          kclm.ce.bitfileid,
920 2          cjm.ce.mode,
921 2          namesize,
922 2          {
923 2              namesize > 0 ? fname: NULL,
924 2              directives_size,
925 2              directives_list,
926 2              /* directive_size */
927 2              cjm.ce.filesize,
928 2              directives */
929 2              prev_abd))
930 2      {
931 2          0, *** warning: could not submit marked file for restore.*
932 2      }

```

```

931 2      * skipping file %s.", filename);
932 2    }
933 1    return 0;
934 1
935 1    memory(prev_abd, kabd, sizeof(abda_wid_t));
936 1
937 1    #if 0
938 1        /* G.
939 1        Sachari: since buf is uninitialized and function is #if 0'd */
940 1        push_to_submitfile(buf, strlen(buf));
941 1        kwrite;
942 1    return 1;
943 1
944 1    /*
945 1    * Creation and destruction of vol'd's now take place
946 1    * outside of "go" command.
947 1    *
948 1    * now add bitfile id to volumes needed report
949 1    * if (rcx->abvlok)
950 1    * {
951 1        rcx->abvlist = abv1_genvol_idlist(
952 1        1,
953 1        rcx->abvlist, &lm_cg_bitfileid,
954 1        &abp_status);
955 1    * }
956 1    *
957 1    * fprintf(
958 1    * stderr, "Unable to generate volumes needed report. %s (%d)",
959 1    * e_get_error_text(&rcx_status), &rcx_status);
960 1    * rcx->abvlok = FALSE;
961 1    * }
962 1    /* end of push_binfo_to_submitfile() */
963 1    }

```

```

964 /* Returns: -1 for error encountered
965             otherwise number of bytes written
966 */
967 *
968 *
969 */
970 static int
971 push_to_submiffle(int fd,
972                  char *buf,
973                  uint_t nbytes)
974 {
975     if (0
976         int wrote;
977         int save_errno;
978         if (debugmode)
979             {
980                 (void)write(fileno(stdout), buf, nbytes);
981             }
982             wrote = loopwrite(fd, buf, (int)nbytes, write_no_eintr);
983             save_errno = errno;
984             if (wrote != (int)nbytes)
985                 /* short write error
986                  */
987                 /* The log states(0, "vrt" trouble writing submiffle."):
988                    rbe_log_stats(0, "*** wanted to write %d, wrote %d", nbytes, wrote);
989                    */
990                 }
991                 error = save_errno;
992                 return wrote;
993             }
994             return nbytes;
995             /* end of push_to_submiffle() */
996 }
997
998
999
1000

```

```

1002 /*
1003  * Convert an EBS ID to string form.  Slide leading zeros.
1004  */
1005
1006 static void
1007 ehbfsdarc_Lz(ebfs_uid_t *ehbfsdp,
1008              register char *buf)
1009 {
1010     register char *p;
1011     register char *q;
1012     unsigned long longvals[4];
1013     int i;
1014     int j;
1015     char *hexdigits = "0123456789abcdef";
1016
1017     memcpy(longvals, ehbfsdp, 16);
1018
1019     q = tmpbuf;
1020
1021     for (i = 0; i < 4; i++)
1022     {
1023         int j;
1024         register unsigned long ul;
1025         q += 8;
1026         ul = longvals[i];
1027
1028         for (j = 0; j < 8; j++)
1029         {
1030             /* q = hexdigits[ LONG2INT(ul & 0xf) ];
1031                ul >= 4;
1032             }
1033             q += 8;
1034         }
1035
1036         tmpbuf[32] = '\0';
1037
1038         /*
1039          * Skip over leading 0 characters
1040          */
1041         for (q = tmpbuf; *q == '0'; q++)
1042         {
1043             /* null */
1044         }
1045
1046         /*
1047          * Copy the rest, up to and including the comma, to output buf
1048          */
1049         p = buf;
1050         while ((*p++ = *q++) != '\0')
1051         {
1052             /* null */
1053         }
1054         /* end of ehbfsdarc_Lz() */
1055     }
1056 }
1057
1058
1059
1060

```

```

1061 static int
1062 ssaid2ebid(ssaid_t *ssaid,
1063            ebfs_uid_t *abidp)
1064 {
1065     ibsaeset_t ss;
1066     errno
1067
1068     /*
1069      * clobber it, to make failure to find match obvious
1070      */
1071     (void)memset((char *)abidp, 0, sizeof *abidp);
1072
1073     if (ssr == ss_find(ssaid, &ss, 0) == 0)
1074         memory(ebidp, &ss.ss_dirid, sizeof(ebfs_uid_t));
1075 }
1076
1077 return err ? -1 : 0;
1078
1079 /* end of ssaid2ebid() */
1080 }

```

```

1086 /*****
1087 ** Routine: RSTSL_get_catalog_info
1088 **
1089 **
1090 ** Inputs: time - The time of the backup that is being worked
1091 **           with
1092 **
1093 ** Outputs: level - reference to the level string to be output
1094 **           numrec - reference to the level of the backup
1095 **           catType - reference to the string which will contain the
1096 **                    number
1097 **           ofRecords - reference to the string which will contain the
1098 **                    number
1099 **           of catalog for the specified backup
1100 **           type
1101 **
1102 ** Purpose: Function to retrieve backup level, catalog type, and number of
1103 **           records and then return it to the client
1104 **
1105 ** Return Codes: E_SUCCESS - if the catalog exists and able to
1106 **                 return
1107 **                 E_FRL_RECOVER_NO_CATALOG - error getting the catalog
1108 **                 info
1109 **
1110 *****/

```

```

/*
 *errno_t
 *RSPD_get_catalog_info() const time_t time,
 *char **level,
 *char **catType)
 */
/* Called from re_get_catalog_info_svc RPC call */
cat_descriptor *catDescr; /* pointer to the catalog descriptor */
rhead_t rhead; /* pointer to head of a catalog retrieved from
 * the catalog descriptor */
int plane_count=0; /* used to keep track of the traversal through
 * the catalog plane structure */
int number_of_planes; /* total number of planes to traverse */

number_of_planes=(tcp->rc_nplanes); /* must be negated because of
 * previous
 * implementation of
 * cat struct */
do /* traverse the list of planes in the catalog structure */

```


Page 103 of 248	RSTSL_get_catalog_info	Fri Jan 04 16:35:25 2008
1154 2	/*	
1155 2	catchptr = malloc(sizeof(rc_jump, plane_count));	
1156 2	if (NULL == catchptr) /* if there are no catalogs */	
1157 3	{	
1158 3	return (EP_RA_RECOVER_NO_CATALOG);	
1159 2	}	
1160 2	if (0 != catd_get_cat_head(&catchptr, &catptr)) /* if there is no head	
1161 2	/* there is	
1162 2	still not	
1163 2	*catalog	
1164 2	*/	
1165 2	{	
1166 2	return (EP_RA_RECOVER_NO_CATALOG);	
1167 2	}	
1168 2	plane_count--; /* decrement counter, must go into negatives */	
1169 2	/* while the backup we are looking for has not been found, and	
1170 2	* we have not traversed through the entire list	
1171 2	*/	
1172 2	while ((time = catchptr->ch_time) < {	
1173 2	plane_count = number_of_planes);	
1174 2	* must malloc memory	
1175 2	* because cannot	
1176 2	* a ulong	
1177 2	sizeof	
1178 2	*/	
1179 2	sprintf("numrec, %u", catchptr->ch_nitems);	
1180 2	/* make the ulong a string	
1181 2	* which is copied	
1182 2	from the	
1183 2	head of the	
1184 2	catalog	
1185 2	*/	
1186 2	(*level) = (char *)calloc(1, 2);	
1187 2	(*level)[0] = catchptr->ch_level;	
1188 2	switch (catchptr->ch_state)	
1189 2	{	
1190 2	case SSCAT_PARTIAL:	
1191 2	*catype=esl_strdup("PARTIAL");	
1192 2	break;	
1193 2	case SSCAT_UNSORTED:	
1194 2	*catype=esl_strdup("UNSORTED");	
1195 2	break;	
1196 2	case SSCAT_SORTED:	
1197 2	*catype=esl_strdup("SORTED");	
1198 2	break;	
1199 2	case SSCAT_NULL:	
1200 2	*catype=esl_strdup("FULL");	
1201 2	break;	
1202 2	case SSCAT_DELTA:	
1203 2	*catype=esl_strdup("DELTA");	
1204 2	break;	
1205 2	case SSCAT_EXPIRED:	
1206 2	*catype=esl_strdup("EXPIRED");	
1207 2	break;	
1208 2	case SSCAT_NONE:	
1209 2	*catype=esl_strdup("NONE");	

Page 104 of 248	RSTSL_get_catalog_info	Fri Jan 04 16:35:25 2008
1189 2	break;	
1190 2	default:	
1191 2	*catype=esl_strdup("UNKNOWN");	
1192 2	break;	
1193 2	}	
1194 2	return E_SUCCESS;	
1195 2		
1196 2		

1	#ifdef RESMAIN	(x)	
2	#define LONG2INT(x)		
3	#endif		
4	#define EXTENSIONS	/* needed by eb headers: ebu11 ? */	
5			
6	#include <stdio.h>		
7	#include <string.h>		
8	#include <unistd.h> (const char *)		
9	#include <sys/types.h>		
10	#include <stdlib.h>		
11	#include <ctype.h>		
12	#include <sys/types.h>		
13	#include <sys/stat.h>		
14	#include <fcntl.h>		
15	#include <limits.h>		
16	#include <sys/portable.h>		
17	#include <util.h>		
18	#include <sys/param.h>		
19	#include <sys/ebfs.h>		
20			
21	#include <restore/EBFSSubmitApi.h>		
22	#include <submitfile.h>		
23			
24	extern int		
25	GetSubmitFile(int ID, int elementID, char *buf, int maxsize,		
26	int *status)		
27	extern int		
28	SetSubmitFile(int ID, int elementID, char *submit_file, int *status)		
29			
30			
31	#ifdef RESMAIN		
32	struct submitfile_bitfile_info		
33	{		
34	char ebfs_dir[33];		
35	char ebfs_file[33];		
36	boolean_t is_bitfile_dir;		
37	int file_renamed_size; /*needed for Netware */		
38	char *file_renamed; /*NULL if no rename*/		
39	int directive_size;		
40	char *directives;		
41	ulong file_size_high;		
42	ulong file_size_low;		
43	};		
44			
45	char *testmain_filename_G = NULL;		
46			
47	#endif		
48			
49	/*		
50	*/		
51	/* array to convert ascii hex character to hex value		
52	*/		
53	uchar_t hdigit_conv[256] =		
54	{		
55			
56	/*		
57	*/		
58	/* ascii characters begin here		
59			
60	0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,		
61	0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,		
62	0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,		
63	0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 0, 0, 0, 0, 0, 0,		
64	0, 10, 11, 12, 13, 14, 15, 0, 0, 0, 0, 0, 0, 0, 0,		
65	0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,		
66	0, 10, 11, 12, 13, 14, 15, 0, 0, 0, 0, 0, 0, 0, 0,		
67	0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,		

67	1	0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,	
68	1	0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,	
69	1	0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,	
70	1	0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,	
71	1	0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,	
72	1	0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,	
73	1	0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,	
74	1	0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,	
75	1	0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,	
76	1	0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,	
77	1	0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,	
78	1	0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,	
79	1	0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,	
80	1	static int	
81	1	writeToFileSubmitFile(const int submit_fd,	
82	1	int *status)	
83	1		
84	1	static void	
85	1	ebfsid2str_1z(const ebfs_id_t *ebfsid,	
86	1	register char *buf);	
87	1		
88	1	int	
89	1	str_1z2ebfsid(char *ebfsidstr, ebfs_id_t *ebfs_id,	
90	1		
91	1	static ssize_t	
92	1	WriteFile(FILE *f, const void *buf, size_t nbyte);	
93	1	static ssize_t	
94	1	ReadFile(FILE *f, void *buf, size_t nbyte);	
95	1		
96	1	/* What about ebfsidstr_1z ?? */	
97	1		
98	1	static char	
99	1	CreateSubmitFileName(int SubmitObjID,	
100	1	int SubmitElemID,	
101	1	int *status)	
102	1		
103	1	{	
104	1	int buffersize = 0;	
105	1	int fd;	
106	1	int counter = 0;	
107	1	boolean_t done = FALSE;	
108	1	int buffersize_max = FALSE;	
109	1	char *submit_file_name = NULL;	
110	1	buffer[0] = 0;	
111	1		
112	1		
113	1	printf(
114	1	buffer, "%s/%s %d %d", SUBMIT_FILE_DIR, SUBMIT_FILE_BASENAME,	
115	1	getpid(), SubmitElemID);	
116	1	while (done_flag == FALSE)	
117	1	{	
118	1	if (--i == access(Buffer, F_OK))	
119	1	{	
120	1	if (ENOENT == errno)	
121	1	{	
122	1	done_flag = TRUE;	
123	1	}	
124	1	else if (EINTR == errno)	
125	1	{	
126	1	continue;	
127	1	}	
128	1		
129	1		
130	1		
131	1	else	
132	1	printf(buffer, "%s/%s %d %d",	
133	1		

Page 111 of 248	CreateSubmitFileName	Fri Jan 04 16:35:25 2008
132 2	SUBMIT_FILE_PATH, SUBMIT_FILE_FILENAME,	
133 2	groupid(), SubmitElemID, counter++);	
134 1	}	
137 1	while (1)	
138 2	{	
139 2	fd = open(buffer, O_WRONLY O_CREAT O_TRUNC, S_IRUSR S_IWUSR);	
141 2	if (fd == -1)	
142 3	{	
143 4	if (errno == EINTR)	
144 4	{	
145 4	continue;	
146 3	}	
147 3	else	
148 4	{	
149 4	*status = errno;	
150 4	return NULL;	
151 3	}	
152 2	}	
154 2	break;	
155 1	}	
157 1	fclose(fd);	
159 1	submit_file_name = (char *) strdup(buffer);	
161 1	if (submit_file_name == NULL)	
162 2	{	
163 2	*status = ENOMEM;	
164 2	return NULL;	
165 1	}	
167 1	return submit_file_name;	
168	}	

Page 112 of 248	OpenSubmitFile	Fri Jan 04 16:35:25 2008
170	/*	
171	* OpensubmitFile()	
172	* No be called to open a submit file. The submitfile	
173	* name will set if this is open for write. If open for	
174	* read the submit file be retrieved in the submitElemID.	
175	* If opened for write the file will be created with write	
176	* and read for the owner (which should be root) and	
177	* open for read only.	
178	* If open for read the file will be set to be read by the	
179	* owner only and the file opened for read only.	
180	* Args:	
181	* (1) boolean cy WriteAccess -- TRUE if opening for write,	
182	* (2) int ID -- for Submit Object ID,	
183	* (3) int elementID -- Submit Element ID,	
184	* (4) int *status -- errno that caused failure.	
185	* Returns: int	
186	* Less than zero for failure.	
187	* zero or greater is the fd of the open submit file.	
188	* zero or greater is the fd of the open submit file.	
189	* zero or greater is the fd of the open submit file.	
190	*/	
191	int	
192	OpenSubmitFile(boolean cy WriteAccess,	
193	int ID,	
194	int elementID,	
195	int *status)	
196 1	{	
197 1	int fd;	
198 1	char *SubmitFileName;	
199 1	int oflag;	
200 1	int mode;	
201 1	int CreateStatus;	
202 1	int SetStatus;	
203 1	int SubmitObjectID = ID;	
204 1	int SubmitElemID = elementID;	
206 1	char SubmitFileBuf[B_PATH_MAX];	
208 1	/* Get the submit file name */	
210 1	if(TRUE == WriteAccess)	
211 2	{	
212 2	int CreateStatus;	
213 2	int SetStatus;	
214 2	if(NULL == (SubmitFileName = CreateSubmitFileName(SubmitObjectID,	
215 2	SubmitElemID,	
216 2	CreateStatus)))	
217 2	{	
218 3	return -1;	
219 3	}	
220 2	#ifdef TESTMAIN	
221 2	testmain_filename_G = strdup(SubmitFileName);	
222 2		
223 2	else	
224 2	if(0 != SetSubmitFile(SubmitObjectID,	
225 2	SubmitElemID,	
226 2	SubmitFileName,	
227 2	CreateStatus))	
228 2	{	
229 3	return -1;	
230 3	}	

```

233 2      }
235 2      fcntl(f
237 2          oflag = O_WRONLY|O_TRUNC|O_CREAT,
238 2          mode = S_IRWRITE|S_IRREAD;
239 1      ) else /* Read means the file exists, lets get the name */
240 1      {
241 1          int GetStatus;
242 2      }
243 2      #ifdef TESTMAIN
245 2      strcpy(SubmitFileBuf, testmain_filename.G);
247 2      #else
249 2          if(0 != GetCSSubmitFile(SubmitObjectID,
250 2          SubmitFileMD,
251 2          SubmitFileBuf, E_PATH_MAX,
252 2          &GetStatus))
253 1          {
254 1              return -1;
255 2          }
257 2      #endif
259 2      SubmitFileName = SubmitFileBuf;
260 2      oflag = O_RDONLY;
261 2      mode = S_IRREAD;
262 1      }
263 1      do
264 1      {
265 2          fd = open(SubmitFileName, oflag, mode); /* No Body */
267 2          while((-1 == fd) && (EINVAL == errno));
268 1      }
269 1      if((fd < 0)
270 1          *status = errno;
271 2          return -1;
272 1      )
273 1      /* Free is if was malloc'ed */
274 1      if((NULL != SubmitFileName) &&
275 1          (SubmitFileBuf != SubmitFileName))
276 1      {
277 1          free(SubmitFileName);
278 1      }
279 1      return fd;
280 1      }
281 1      }
282 1      }
283 1      }
284 1      }
285 1      }
286 1      }

```

```

289 1      /*
290 1      * CloseSubmitFile()
291 1      * To be called to close a submit file. The submitfile
292 1      * will be simply closed if the file was read only (see
293 1      * WriteAccess arg). If the close is for a file opened for
294 1      * write then a trailer will be appended to the submit
295 1      * file and the file will be set to owner read permission
296 1      * only and the file will be closed.
297 1      * Args:
298 1      * (I) int submit_fd -- the submit file to close.
299 1      * (I) bool* by WriteAccess -- TRUE if opening for write.
300 1      * (O) int *status -- errno that caused failure.
301 1      * Returns: int
302 1      * 0 for success
303 1      * non-zero for failure, check status.
304 1      */
305 1      *
306 1      *
307 1      */
308 1      *
309 1      *
310 1      int CloseSubmitFile(int submit_fd,
311 1      bool* byWriteAccess,
312 1      int *status)
313 1      {
314 1          int temp_status;
315 1          if(TRUE == WriteAccess)
316 1          {
317 1              /* If this close is for a write access,
318 1              * then append a trailer to the submit file
319 1              * And tighten up the permissions.
320 1              */
321 1              if(0 != WriteTrailerToSubmitFile(submit_fd, status))
322 1              {
323 1                  return -1;
324 1              }
325 1              while((-1 == (temp_status = fchmod(submit_fd, O_RDONLY))) &&
326 1                  EINTR == errno)
327 1              { /* no body */
328 1              }
329 1              if(-1 == temp_status)
330 1              {
331 1                  *status = errno;
332 1                  return -1;
333 1              }
334 1              while((-1 == (temp_status = close(submit_fd))) &&
335 1                  EINTR == errno)
336 1              {
337 1              }
338 1              if(-1 == temp_status)
339 1              {
340 1                  *status = errno;
341 1                  return -1;
342 1              }
343 1              if(-1 == temp_status)
344 1              {
345 1                  *status = errno;
346 1                  return -1;
347 1              }
348 1              if(-1 == temp_status)
349 1              {
350 1                  *status = 0;
351 1                  return 0;
352 1              }
353 1              }
354 1              }
355 1              }
356 1              }
357 1              }
358 1              }
359 1              }
360 1              }
361 1              }
362 1              }
363 1              }
364 1              }
365 1              }
366 1              }
367 1              }
368 1              }
369 1              }
370 1              }
371 1              }
372 1              }
373 1              }
374 1              }
375 1              }
376 1              }
377 1              }
378 1              }
379 1              }
380 1              }
381 1              }
382 1              }
383 1              }
384 1              }
385 1              }
386 1              }
387 1              }
388 1              }
389 1              }
390 1              }
391 1              }
392 1              }
393 1              }
394 1              }
395 1              }
396 1              }
397 1              }
398 1              }
399 1              }
400 1              }
401 1              }
402 1              }
403 1              }
404 1              }
405 1              }
406 1              }
407 1              }
408 1              }
409 1              }
410 1              }
411 1              }
412 1              }
413 1              }
414 1              }
415 1              }
416 1              }
417 1              }
418 1              }
419 1              }
420 1              }
421 1              }
422 1              }
423 1              }
424 1              }
425 1              }
426 1              }
427 1              }
428 1              }
429 1              }
430 1              }
431 1              }
432 1              }
433 1              }
434 1              }
435 1              }
436 1              }
437 1              }
438 1              }
439 1              }
440 1              }
441 1              }
442 1              }
443 1              }
444 1              }
445 1              }
446 1              }
447 1              }
448 1              }
449 1              }
450 1              }
451 1              }
452 1              }
453 1              }
454 1              }
455 1              }
456 1              }
457 1              }
458 1              }
459 1              }
460 1              }
461 1              }
462 1              }
463 1              }
464 1              }
465 1              }
466 1              }
467 1              }
468 1              }
469 1              }
470 1              }
471 1              }
472 1              }
473 1              }
474 1              }
475 1              }
476 1              }
477 1              }
478 1              }
479 1              }
480 1              }
481 1              }
482 1              }
483 1              }
484 1              }
485 1              }
486 1              }
487 1              }
488 1              }
489 1              }
490 1              }
491 1              }
492 1              }
493 1              }
494 1              }
495 1              }
496 1              }
497 1              }
498 1              }
499 1              }
500 1              }
501 1              }
502 1              }
503 1              }
504 1              }
505 1              }
506 1              }
507 1              }
508 1              }
509 1              }
510 1              }
511 1              }
512 1              }
513 1              }
514 1              }
515 1              }
516 1              }
517 1              }
518 1              }
519 1              }
520 1              }
521 1              }
522 1              }
523 1              }
524 1              }
525 1              }
526 1              }
527 1              }
528 1              }
529 1              }
530 1              }
531 1              }
532 1              }
533 1              }
534 1              }
535 1              }
536 1              }
537 1              }
538 1              }
539 1              }
540 1              }
541 1              }
542 1              }
543 1              }
544 1              }
545 1              }
546 1              }
547 1              }
548 1              }
549 1              }
550 1              }
551 1              }
552 1              }
553 1              }
554 1              }
555 1              }
556 1              }
557 1              }
558 1              }
559 1              }
560 1              }
561 1              }
562 1              }
563 1              }
564 1              }
565 1              }
566 1              }
567 1              }
568 1              }
569 1              }
570 1              }
571 1              }
572 1              }
573 1              }
574 1              }
575 1              }
576 1              }
577 1              }
578 1              }
579 1              }
580 1              }
581 1              }
582 1              }
583 1              }
584 1              }
585 1              }
586 1              }
587 1              }
588 1              }
589 1              }
590 1              }
591 1              }
592 1              }
593 1              }
594 1              }
595 1              }
596 1              }
597 1              }
598 1              }
599 1              }
600 1              }
601 1              }
602 1              }
603 1              }
604 1              }
605 1              }
606 1              }
607 1              }
608 1              }
609 1              }
610 1              }
611 1              }
612 1              }
613 1              }
614 1              }
615 1              }
616 1              }
617 1              }
618 1              }
619 1              }
620 1              }
621 1              }
622 1              }
623 1              }
624 1              }
625 1              }
626 1              }
627 1              }
628 1              }
629 1              }
630 1              }
631 1              }
632 1              }
633 1              }
634 1              }
635 1              }
636 1              }
637 1              }
638 1              }
639 1              }
640 1              }
641 1              }
642 1              }
643 1              }
644 1              }
645 1              }
646 1              }
647 1              }
648 1              }
649 1              }
650 1              }
651 1              }
652 1              }
653 1              }
654 1              }
655 1              }
656 1              }
657 1              }
658 1              }
659 1              }
660 1              }
661 1              }
662 1              }
663 1              }
664 1              }
665 1              }
666 1              }
667 1              }
668 1              }
669 1              }
670 1              }
671 1              }
672 1              }
673 1              }
674 1              }
675 1              }
676 1              }
677 1              }
678 1              }
679 1              }
680 1              }
681 1              }
682 1              }
683 1              }
684 1              }
685 1              }
686 1              }
687 1              }
688 1              }
689 1              }
690 1              }
691 1              }
692 1              }
693 1              }
694 1              }
695 1              }
696 1              }
697 1              }
698 1              }
699 1              }
700 1              }
701 1              }
702 1              }
703 1              }
704 1              }
705 1              }
706 1              }
707 1              }
708 1              }
709 1              }
710 1              }
711 1              }
712 1              }
713 1              }
714 1              }
715 1              }
716 1              }
717 1              }
718 1              }
719 1              }
720 1              }
721 1              }
722 1              }
723 1              }
724 1              }
725 1              }
726 1              }
727 1              }
728 1              }
729 1              }
730 1              }
731 1              }
732 1              }
733 1              }
734 1              }
735 1              }
736 1              }
737 1              }
738 1              }
739 1              }
740 1              }
741 1              }
742 1              }
743 1              }
744 1              }
745 1              }
746 1              }
747 1              }
748 1              }
749 1              }
750 1              }
751 1              }
752 1              }
753 1              }
754 1              }
755 1              }
756 1              }
757 1              }
758 1              }
759 1              }
760 1              }
761 1              }
762 1              }
763 1              }
764 1              }
765 1              }
766 1              }
767 1              }
768 1              }
769 1              }
770 1              }
771 1              }
772 1              }
773 1              }
774 1              }
775 1              }
776 1              }
777 1              }
778 1              }
779 1              }
780 1              }
781 1              }
782 1              }
783 1              }
784 1              }
785 1              }
786 1              }
787 1              }
788 1              }
789 1              }
790 1              }
791 1              }
792 1              }
793 1              }
794 1              }
795 1              }
796 1              }
797 1              }
798 1              }
799 1              }
800 1              }
801 1              }
802 1              }
803 1              }
804 1              }
805 1              }
806 1              }
807 1              }
808 1              }
809 1              }
810 1              }
811 1              }
812 1              }
813 1              }
814 1              }
815 1              }
816 1              }
817 1              }
818 1              }
819 1              }
820 1              }
821 1              }
822 1              }
823 1              }
824 1              }
825 1              }
826 1              }
827 1              }
828 1              }
829 1              }
830 1              }
831 1              }
832 1              }
833 1              }
834 1              }
835 1              }
836 1              }
837 1              }
838 1              }
839 1              }
840 1              }
841 1              }
842 1              }
843 1              }
844 1              }
845 1              }
846 1              }
847 1              }
848 1              }
849 1              }
850 1              }
851 1              }
852 1              }
853 1              }
854 1              }
855 1              }
856 1              }
857 1              }
858 1              }
859 1              }
860 1              }
861 1              }
862 1              }
863 1              }
864 1              }
865 1              }
866 1              }
867 1              }
868 1              }
869 1              }
870 1              }
871 1              }
872 1              }
873 1              }
874 1              }
875 1              }
876 1              }
877 1              }
878 1              }
879 1              }
880 1              }
881 1              }
882 1              }
883 1              }
884 1              }
885 1              }
886 1              }
887 1              }
888 1              }
889 1              }
890 1              }
891 1              }
892 1              }
893 1              }
894 1              }
895 1              }
896 1              }
897 1              }
898 1              }
899 1              }
900 1              }
901 1              }
902 1              }
903 1              }
904 1              }
905 1              }
906 1              }
907 1              }
908 1              }
909 1              }
910 1              }
911 1              }
912 1              }
913 1              }
914 1              }
915 1              }
916 1              }
917 1              }
918 1              }
919 1              }
920 1              }
921 1              }
922 1              }
923 1              }
924 1              }
925 1              }
926 1              }
927 1              }
928 1              }
929 1              }
930 1              }
931 1              }
932 1              }
933 1              }
934 1              }
935 1              }
936 1              }
937 1              }
938 1              }
939 1              }
940 1              }
941 1              }
942 1              }
943 1              }
944 1              }
945 1              }
946 1              }
947 1              }
948 1              }
949 1              }
950 1              }
951 1              }
952 1              }
953 1              }
954 1              }
955 1              }
956 1              }
957 1              }
958 1              }
959 1              }
960 1              }
961 1              }
962 1              }
963 1              }
964 1              }
965 1              }
966 1              }
967 1              }
968 1              }
969 1              }
970 1              }
971 1              }
972 1              }
973 1              }
974 1              }
975 1              }
976 1              }
977 1              }
978 1              }
979 1              }
980 1              }
981 1              }
982 1              }
983 1              }
984 1              }
985 1              }
986 1              }
987 1              }
988 1              }
989 1              }
990 1              }
991 1              }
992 1              }
993 1              }
994 1              }
995 1              }
996 1              }
997 1              }
998 1              }
999 1              }
1000 1              }

```

Page 116 of 248
352
}

Page 116 of 248

```
355  /*  
356  * Internal use only.  
357  */  
358  static int  
359  WriteTailorToSubmitFile(const int submit_fd,  
360                          int *status)  
361  {  
362      if(! Write(submit_fd, "!", 1))  
363      {  
364          *status = errno;  
365          return -1;  
366      }  
367      return 0;  
368  }
```

Page 116 of 248

./libx_restoreSubmitFile.c 7

Fri Jan 04 16:35:25 2008

Page 116 of 248

./libx_restoreSubmitFile.c 8

Fri Jan 04 16:35:25 2008


```

489 1   ret_write = write(subm_c_fd, buf, strlen(buf));
490 2   if (ret_write != strlen(buf))
491 3   {
492 4       *return -1;
493 5   }
494 6   return 0;
495 7   }
496 8   }

```

```

499   static ssize_t
500   Write(int fd, const void *buf, size_t nbyte)
501   {
502   1   int ret_write;
503   2   while(-1 == (ret_write = write(fd, buf, nbyte))) &&
504   3       (errno == EINTR || EAGAIN == errno)
505   4   {
506   5       return ret_write;
507   6   }

```


Page 123 of 248	ReadBinfileInFromSubmitFile	Fri Jan 04 16:35:25 2008
572 1	char temp_look;	
573 1	int ret_read;	
574 1	int index;	
575 1	boolean ly have_read, filesize = FALSE;	
576 1	boolean ly have_not_started, filesize = FALSE;	
577 1	boolean ly directive_read = FALSE;	
578 1	boolean ly remained_read = FALSE;	
579 1		
580 2	memset(temp_read_buf_ptr, 0, 5000);	
581 1	memset(temp_ebfs_str_ptr, 0, 34);	
582 1	ret_read = Read(submit_fd, &temp_look, 1);	
583 1	if (! ret_read)	
584 1	{	
585 1	return -1;	
586 2	}	
587 1	if (ret_read == temp_look)	
588 1	{	
589 2	/* This is an end of file condition */	
590 2	return 0;	
591 2	else if (ret_read == temp_look)	
592 2	{	
593 2	memset(ebfs_dir_ptr, prev_ebfs_dir_ptr, sizeof(ebfs_dir_ptr));	
594 1	else if (ret_read == temp_look)	
595 2	{	
596 2	temp_ebfs_str_ptr[0] = temp_look;	
597 1	temp_ebfs_str_ptr++;	
598 1	/* Now read the next 31 plus the comma */	
599 2	ret_read = Read(submit_fd, temp_read_buf_ptr, 32);	
600 2	if ((32 != ret_read) (ret_read != temp_read_buf_ptr[31]))	
601 2	{	
602 2	return -1;	
603 2	For(index = 0; index < 31; index++, temp_ebfs_str_ptr++)	
604 2	{	
605 2	if(isxdigit(temp_read_buf_ptr[index]))	
606 2	{	
607 2	*temp_ebfs_str_ptr = temp_read_buf_ptr[index];	
608 2	else	
609 2	{	
610 2	return -1;	
611 2	}	
612 2	temp_ebfs_str_ptr = temp_ebfs_str_ptr;	
613 2	strncpy(temp_ebfs_str_ptr, ebfs_dir_ptr, sizeof(ebfs_dir_ptr));	
614 2	memset(temp_ebfs_str_ptr, 0, 5000);	
615 2	else	
616 2	{	
617 2	return -1;	
618 2	}	
619 2	temp_ebfs_str_ptr = temp_ebfs_str_ptr;	
620 2	strncpy(temp_ebfs_str_ptr, ebfs_dir_ptr, sizeof(ebfs_dir_ptr));	
621 2	memset(temp_ebfs_str_ptr, 0, 5000);	
622 2	else	
623 2	{	
624 2	return -1;	
625 2	}	
626 2	temp_read_buf_ptr = temp_read_buf_ptr;	
627 2	temp_ebfs_str_ptr = temp_ebfs_str_ptr;	
628 2	memset(temp_read_buf_ptr, 0, 34);	
629 2	memset(temp_ebfs_str_ptr, 0, 34);	
630 2	/* the biff file id(32) + the comma + file type char */	
631 2	ret_read = Read(submit_fd, temp_read_buf_ptr, 34);	
632 2		

Page 124 of 248	ReadBinfileInFromSubmitFile	Fri Jan 04 16:35:25 2008
633 1	if ((34 != ret_read) (ret_read != temp_read_buf_ptr[32]))	
634 1	{	
635 1	return -1;	
636 1	}	
637 1	for(index = 0; index < 32; index++, temp_ebfs_str_ptr++)	
638 1	{	
639 1	if(isxdigit(temp_read_buf_ptr[index]))	
640 1	{	
641 1	*temp_ebfs_str_ptr = temp_read_buf_ptr[index];	
642 1	else	
643 1	{	
644 1	return -1;	
645 1	}	
646 1	temp_ebfs_str_ptr = temp_ebfs_str_ptr;	
647 1	strncpy(temp_ebfs_str_ptr, ebfs_dir_ptr, sizeof(ebfs_dir_ptr));	
648 1	if(temp_read_buf_ptr[33] == 'd')	
649 1	{	
650 1	*is_bfile_dir = TRUE;	
651 1	else if (temp_read_buf_ptr[33] == '0')	
652 1	{	
653 1	*is_bfile_dir = FALSE;	
654 1	else	
655 1	{	
656 1	return -1;	
657 1	}	
658 1	temp_read_buf_ptr = temp_read_buf_ptr;	
659 1	temp_ebfs_str_ptr = temp_ebfs_str_ptr;	
660 1	memset(temp_ebfs_str_ptr, 0, 34);	
661 1	memset(temp_attr_size_ptr, 0, 5);	
662 1	memset(temp_file_size_ptr, 0, 17);	
663 1	/* We read 5 bytes here because it will be either	
664 1	* 2) remained directive indicator and length of the directive.	
665 1	* 3) the first 5 characters of the file size.	
666 1	*/	
667 1	while(have_read, filesize)	
668 1	{	
669 1	memset(temp_read_buf_ptr, 0, 5000);	
670 1	ret_read = Read(submit_fd, temp_read_buf_ptr, 5);	
671 1	if (5 != ret_read)	
672 1	{	
673 1	return -1;	
674 1	}	
675 1	if((ret_read == temp_read_buf_ptr[0]) (ret_read == temp_read_buf_ptr[0]))	
676 1	{	
677 1	unsigned long attr_size = 0;	
678 1	char *attr_temp;	
679 1	strncpy(temp_attr_size_ptr, temp_read_buf_ptr + 1, 4);	
680 1	attr_size = atoi(temp_attr_size_ptr, (char**) NULL, 16);	
681 1	attr_temp = (char *) calloc(1, (attr_size + 1));	
682 1	if(NULL == attr_temp)	
683 1	{	
684 1	return -1;	
685 1	}	

```

704 4         return -1;
705 3     }
706 3     reRead = Read(submit_fd, attr_temp, attr_size);
707 3     if (attr_size != reRead)
708 4     {
709 4         return -1;
710 3     }
711 3     if ('v' == temp_read_buf_ptr[0])
712 4     {
713 4         renamed_read = TRUE;
714 4         *renamed_file_size = attr_size;
715 4         *file_renamed = attr_temp;
716 4         continue;
717 4     }
718 3     if ('D' == temp_read_buf_ptr[0])
719 4     {
720 4         directive_read = TRUE;
721 4         *directive_size = attr_size;
722 4         *directives = attr_temp;
723 4         continue;
724 4     }
725 3     }
726 3     )
727 2     else if (',' == temp_read_buf_ptr[0])
728 2     {
729 2         memory[temp_file_size_ptr, &temp_read_buf_ptr[1], 4];
730 2         temp_file_size_ptr += 4;
731 3     }
732 3     /* If the file size is a long then the 5 byte should be '\n'.
733 3     */
734 3     reRead = Read(submit_fd, temp_file_size_ptr, 5);
735 3     if (5 != reRead)
736 4     {
737 4         return -1;
738 3     }
739 3     if (temp_file_size_ptr[4] == '\n')
740 3     {
741 4         temp_file_size_ptr[4] = '\0';
742 4         temp_file_size_ptr = temp_file_size_ptr + 1;
743 4         for (index = 0; index < 8; index++)
744 4         {
745 4             if (isxdigit(temp_file_size_ptr[index]))
746 4             {
747 4                 *filesize = ul_to_h(strtol(temp_file_size_ptr,
748 4                 char **, NULL, 16));
749 4                 have_read_filesize = TRUE;
750 3             }
751 3         }
752 3     }
753 4     else if (isxdigit(temp_file_size_ptr[4]))
754 4     {
755 4         temp_file_size_ptr = &temp_file_size_ptr[9]; /* aton */
756 4         temp_file_size_ptr[4] = '\0';
757 4         reRead = Read(submit_fd, temp_file_size_ptr, 8);
758 4         if (8 != reRead) || (('v' == temp_file_size_ptr[7]))
759 4         {
760 4             return -1;
761 3         }
762 3     }
763 4     temp_file_size_ptr = temp_file_size_ptr + 1;
764 4     temp_file_size_ptr[16] = '\0';
765 4     for (index = 0; index < 16; index++)
766 4     {
767 4         if (isxdigit(temp_file_size_ptr[index]))
768 4         {

```

```

769 6         {
770 6             return -1;
771 6         }
772 4         /* hack lets use a large enough buffer
773 4         to put a "0x" at the beginning of our hexadecimal uhyper
774 4         string */
775 4         memset(temp_abfs_str_buffer, 0, 34);
776 4         temp_abfs_str_buffer[0] = '\0';
777 4         temp_abfs_str_buffer[1] = '\x';
778 4         temp_abfs_str_buffer[2] = '\0';
779 4         strncat(temp_abfs_str_buffer(2), temp_file_size_ptr,
780 4         filesize);
781 4         if (0 != strncat(temp_u_hyper(temp_abfs_str_buffer, filesize))
782 4         {
783 4             return -1;
784 4         }
785 4         have_read_filesize = TRUE;
786 4     }
787 3     )
788 4     {
789 4         return -1;
790 3     }
791 3     }
792 3     }
793 3     }
794 3     }
795 3     }
796 3     }
797 3     }
798 3     }
799 3     }
800 3     }
801 3     }
802 3     }
803 3     }
804 3     }
805 3     }
806 3     }
807 3     }
808 3     }
809 3     }
810 3     }
811 3     }
812 3     }
813 3     }
814 3     }
815 3     }
816 3     }
817 3     }
818 3     }
819 3     }
820 3     }
821 3     }
822 3     }
823 3     }
824 3     }
825 3     }
826 3     }
827 3     }
828 3     }
829 3     }
830 3     }
831 3     }
832 3     }
833 3     }
834 3     }
835 3     }
836 3     }
837 3     }
838 3     }
839 3     }
840 3     }
841 3     }
842 3     }
843 3     }
844 3     }
845 3     }
846 3     }
847 3     }
848 3     }
849 3     }
850 3     }
851 3     }
852 3     }
853 3     }
854 3     }
855 3     }
856 3     }
857 3     }
858 3     }
859 3     }
860 3     }
861 3     }
862 3     }
863 3     }
864 3     }
865 3     }
866 3     }
867 3     }
868 3     }
869 3     }
870 3     }
871 3     }
872 3     }
873 3     }
874 3     }
875 3     }
876 3     }
877 3     }
878 3     }
879 3     }
880 3     }
881 3     }
882 3     }
883 3     }
884 3     }
885 3     }
886 3     }
887 3     }
888 3     }
889 3     }
890 3     }
891 3     }
892 3     }
893 3     }
894 3     }
895 3     }
896 3     }
897 3     }
898 3     }
899 3     }
900 3     }
901 3     }
902 3     }
903 3     }
904 3     }
905 3     }
906 3     }
907 3     }
908 3     }
909 3     }
910 3     }
911 3     }
912 3     }
913 3     }
914 3     }
915 3     }
916 3     }
917 3     }
918 3     }
919 3     }
920 3     }
921 3     }
922 3     }
923 3     }
924 3     }
925 3     }
926 3     }
927 3     }
928 3     }
929 3     }
930 3     }
931 3     }
932 3     }
933 3     }
934 3     }
935 3     }
936 3     }
937 3     }
938 3     }
939 3     }
940 3     }
941 3     }
942 3     }
943 3     }
944 3     }
945 3     }
946 3     }
947 3     }
948 3     }
949 3     }
950 3     }
951 3     }
952 3     }
953 3     }
954 3     }
955 3     }
956 3     }
957 3     }
958 3     }
959 3     }
960 3     }
961 3     }
962 3     }
963 3     }
964 3     }
965 3     }
966 3     }
967 3     }
968 3     }
969 3     }
970 3     }
971 3     }
972 3     }
973 3     }
974 3     }
975 3     }
976 3     }
977 3     }
978 3     }
979 3     }
980 3     }
981 3     }
982 3     }
983 3     }
984 3     }
985 3     }
986 3     }
987 3     }
988 3     }
989 3     }
990 3     }
991 3     }
992 3     }
993 3     }
994 3     }
995 3     }
996 3     }
997 3     }
998 3     }
999 3     }
1000 3     }

```

```

803  /*
804  */
805  int
806  str_12abbsid(char *ebfsdirstr, ebfs_uid_t *ebfs_p)
807  {
808      int index;
809      unsigned int *upptr = (unsigned int *)ebfs_p;
810
811      memset(ebfs_p, 0, 32);
812
813      for(index=0; index < 32; index++)
814      {
815          if(!strcmp(ebfsdirstr[index]))
816              return -1;
817      }
818      else
819      {
820          if((index != 0) && (0 == (index % 8)))
821          {
822              upptr++;
823          }
824          *upptr = *upptr << 4;
825          *upptr = *upptr + htonl_conv(ebfsdirstr[index]);
826      }
827      return 0;
828  }
829  /* end of str_12abbsid */

```

```

833  /*
834  */
835  static void
836  ebfsdirstr_12(const ebfs_uid_t *ebfs_p,
837                register char *buf)
838  {
839      register char *p;
840      register char *q;
841      unsigned long longvals[4];
842      int i;
843      char tmpbuf[32];
844      char *headigs = "0123456789abcdec";
845
846      memcpy(longvals, ebfs_p, 16);
847
848      q = tmpbuf;
849      q = tmpbuf;
850      for (i = 0; i < 4; i++)
851      {
852          int j;
853          register unsigned long ul;
854          q += 8;
855          ul = longvals[i];
856          for (j = 0; j < 8; j++)
857          {
858              *--q = headigs[(LONG2INT(ul & 0xf))];
859              ul >>= 4;
860          }
861          q += 8;
862      }
863      tmpbuf[32] = '\0';
864
865      /*
866      * Skip over leading 0 characters
867      */
868      for (q = tmpbuf; *q == '\0'; q++)
869      ;
870
871      /*
872      * Copy the rest, up to and including the comma, to output buf
873      */
874      p = buf;
875      p = buf;
876      while ((*p++ = *q++) != '\0')
877      {
878          /* null */
879      }
880      /* end of ebfsdirstr_12 */

```

```

889 1 {
890 1     int index;
891 1     unsigned int *uptr = (unsigned int *)ebfsidp;
892 1     char *buf_ptr = buf;
893 1     char *hexdigits = "0123456789abcdef";
894 1
895 1     for(index=0; 4 > index ; index++, uptr++)
896 1     {
897 1         buf_ptr[0] = hexdigits[ (*uptr & 0xf0000000) >> 28 ];
898 2         hexdigits[ (*uptr & 0xf0000000) >> 24 ];
899 2         buf_ptr[1] = hexdigits[ (*uptr & 0x0f000000) >> 20 ];
900 2         buf_ptr[2] = hexdigits[ (*uptr & 0x00f00000) >> 16 ];
901 2         buf_ptr[3] = hexdigits[ (*uptr & 0x000f0000) >> 12 ];
902 2         buf_ptr[4] = hexdigits[ (*uptr & 0x0000f000) >> 8 ];
903 2         buf_ptr[5] = hexdigits[ (*uptr & 0x00000f00) >> 4 ];
904 2         buf_ptr[6] = hexdigits[ (*uptr & 0x000000f0) >> 0 ];
905 2         buf_ptr[7] = hexdigits[ (*uptr & 0x0000000f) >> 0 ];
906 2
907 2     }
908 1     buf_ptr += 8;
909 1     return;
910 1 } /* end of ebfsidptr_12() */

```

```

916     /***** Bifile record format *****/
917     D32,B32,T,nilllrrrrrrXlllxxxxxxxsfsfsfs[fsfsfs]in
918     D32 -- is the bifile directory id in 32 hexadecimal ascii chars.
919     If not present the previous bifile directory will be assumed.
920     D32 -- is the bifile id in 32 hexadecimal ascii chars.
921
922     T -- Character to indicate whether the file being restore
923     was a directory or not. This can be 'd' for directory file
924     type or 'f' for a non-directory file type.
925     Optional nilllrrrrrr
926     -----
927     n -- Rename attribute indicator. This is an ascii 'n'.
928
929     1111 -- length of the renamed file name.
930
931     rrrrrrrrr -- the renamed file name.
932
933     Optional Xlllxxxxxxxsfsfsfs
934     -----
935     X -- Extended attributes directive indicator. This is an ascii 'X'.
936
937     1111 -- length of the Extended attributes directive.
938
939     xxxxxxxxxx -- Extended attribute that needs to be inserted into the
940     xcpio stream.
941
942     fsfsfsfsfsfsfs -- The file size in 8 or 16 hexadecimal ascii,
943     along or hyper.
944
945     \n -- New line bifile record delimiter.
946
947     **** End format section *****/
948
949     #ifndef RESTMAIN
950     int
951     bifile_info_cmp(struct submtfile_bifile_info *binfo1,
952     struct submtfile_bifile_info *binfo2)
953     {
954         int temp_cmp;
955
956         if(0 != (temp_cmp = strcmp(
957             binfo1->ebfs_filepath, binfo2->ebfs_filepath)))
958             return temp_cmp;
959
960         if(0 != (temp_cmp = strcmp(
961             binfo1->ebfs_dir, binfo2->ebfs_dir)))
962             return temp_cmp;
963
964         if(0 != (temp_cmp = strcmp(
965             binfo1->ebfs_dir, binfo2->ebfs_dir)))
966             return temp_cmp;
967
968         if(0 != (temp_cmp = strcmp(
969             binfo1->ebfs_dir, binfo2->ebfs_dir)))
970             return temp_cmp;
971
972         if(binfo1->is_bifile_dir != binfo2->is_bifile_dir)
973             return binfo1->is_bifile_dir - binfo2->is_bifile_dir;
974
975         if(binfo1->is_file_renamed_size != binfo2->is_file_renamed_size)
976             return binfo1->is_file_renamed_size - binfo2->is_file_renamed_size;

```


File Jan 04 16:35:25 2008	inbuffer	Page 133 of 246
1066 1	*67890123678901236789012367890123*	
1067 1	*0000456789abcde0123456789abcde*	
1068 1	TRUE, 10, "/dir1/dir2", 0, NULL, 2, 5);	
1070 1	/* 8 */	
1071 1	COPY_BINFO(testbuffer, 8,	
1072 1	*67890123678901236789012367890123*,	
1073 1	*0000456789abcde0123456789abcde*,	
1074 1	FALSE, 10, "/one/two/x", 11, "DIRECT-one", 2, 5);	
1076 1	/* 9 */	
1077 1	COPY_BINFO(testbuffer, 9,	
1078 1	*67890123678901236789012367890123*,	
1079 1	*0000456789abcde0123456789abcde*,	
1080 1	FALSE, 10, "/one/two/x", 11, "DIRECT-one", 2, 5);	
1082 1	/* 10 */	
1083 1	COPY_BINFO(testbuffer, 10,	
1084 1	*67890123678901236789012367890123*,	
1085 1	*0000456789abcde0123456789abcde*,	
1086 1	FALSE, 10, "/one/two/x", 11, "DIRECT-one", 2, 5);	
1088 1	/* 11 */	
1089 1	COPY_BINFO(testbuffer, 11,	
1090 1	*67890123678901236789012367890123*,	
1091 1	*0000456789abcde0123456789abcde*,	
1092 1	FALSE, 10, "/one/two/x", 11, "DIRECT-one", 2, 5);	
1094 1	}	

File Jan 04 16:35:25 2008	main	Page 134 of 246
1096 1	int main()	
1097 1	{	
1098 1	abfs_uid_t bfileId;	
1099 1	abfs_uid_t dirId;	
1100 1	abfs_uid_t prevId;	
1101 1	u_hyper_t fileSize;	
1102 1	int status;	
1103 1	int index;	
1104 1	struct submitFile_bfile_info readbuffer[12];	
1105 1		
1107 1	memset(&prevId, 0, sizeof(abfs_uid_t));	
1109 1	memset(&readbuffer, 0,	
1110 1	12 * sizeof(struct submitFile_bfile_info));	
1112 1	inbuffer(testbuffer);	
1114 1	fd = OpensubmitFile(TRUE, 1, 1, &status);	
1116 1	for(index = 0; index < 12; index++)	
1117 2	{	
1120 2	(void)err_12abfsId(testbuffer[index], abfs_dirp, &dirId);	
1121 2	(void)err_12abfsId(testbuffer[index], abfs_filep, &bfileId);	
1122 2	fileSize_low = testbuffer[index].fileSize_low;	
1123 2	fileSize_high = testbuffer[index].fileSize_high;	
1125 2	if(0 != WritebfileInfoForSubmitFileId,	
1126 2	abfs_dirp,	
1127 2	abfs_bfileId,	
1128 2	testbuffer[index],	
1129 2	testbuffer[index],	
1130 2	file_renamed_size,	
1131 2	file_renamed,	
1132 2	testbuffer[index],	
1133 2	testbuffer[index],	
1134 2	respective_size,	
1135 2	testbuffer[index], &directIev,	
1136 3	fileSize,	
1137 2	&prevId);	
1138 2	}	
1139 2	(void)CloseSubmitFileId,	
1140 2	TRUE,	
1141 2	&status);	
1143 1	fd = OpensubmitFile(FALSE, 1, 1, &status);	
1146 1	memset(&prevId, 0, sizeof(abfs_uid_t));	
1148 1	for(index = 0; index < 12; index++)	
1149 2	{	
1150 2	ReadbfileInfoFromSubmitFileId,	
1151 2	abfs_dirp,	
1152 2	abfs_bfileId,	
1153 2	&readbuffer[index], &dirId,	
1154 2	&readbuffer[index], file_renamed_size,	


```

1155 2      &(readbuffer[index]).file_renamed,
1156 2      &filesize,
1157 2      &(readbuffer[index].directive_size),
1158 2      &(readbuffer[index].directives),
1159 2      &prevdirid);
1160 2
1161 2      ebf&idstr_1z(&dirid, readbuffer[index].ebfa_dirp);
1162 2      ebf&idstr_1z(&bitfileid, readbuffer[index].ebfa_filep);
1163 2      readbuffer[index].filesize_low = filesize_low;
1164 2      readbuffer[index].filesize_high = filesize_high;
1165 2
1166 2      if(0 == bitfile_info_cmp(&readbuffer[index],
1167 2                             &readbuffer[index]))
1168 2      {
1169 2          fprintf(stdout, "\nFile bitfile info matched %d\n", index);
1170 2      }
1171 2      else
1172 2      {
1173 2          fprintf(
1174 2              stdout, "\nFile bitfile info DID NOT match %d\n", index);
1175 2      }
1176 2
1177 2      (void)fcloseSubmFile(fId,
1178 2                          FALSE,
1179 2                          &stratus);
1180 2
1181 2      unlink((const char *)testmain_filename_0);
1182 2      free(testmain_filename_0);
1183 2      testmain_filename_0 = NULL;
1184 2      return 0;
1185 2
1186 2      }

```

```

1  /*
2  ** Copyright 1996,1997 EMC Corporation
3  */
4
5  /* EDMSUBMITAPI.CC
6
7  *
8  * Mission Statement: file that contains an API to manage the Submit
9  * objects.
10
11  * Primary Data Acted On:
12
13  * Compile-Time Options:
14
15  * Basic idea here:
16
17     A few calls to manage the Submit objects. This is
18     used by the Process Manager thread.
19
20     */
21
22 #if defined(lint)
23 static char RCS_id [] = "@(#)SRCFile: EDMSUBMITAPI.CC,v *
24     *Revision: 1.0 $ *
25     *Date: 1997/02/06 20:49:15 $ *
26 #endif
27
28 #include <net/c portable.h>
29 #include <net/ed_xopen.h>
30 #include <net/inout.h>
31
32 #include <stdlib.h>
33 #include <sys/types.h>
34 #include <pthread.h>
35
36 #include <restore/EDMSUBMITAPI.h>
37
38 static unsigned int numberOfSubmitObjects = 0;
39 static RMBinaryTree submitObjects;
40 static pthread_mutex_t G_submitmutex = PTHREAD_MUTEX_INITIALIZER;
41
42 /*****
43  *
44  * Routine: lockSubmitMutex
45  *
46  * Inputs: None
47  *
48  * Outputs: None
49  *
50  * Return Codes:
51  *
52  * Purpose: Lock the mutex for the submit object
53  *
54  *
55  *****/
56
57 static void
58 lockSubmitMutex()
59 {
60     static bool seen_ly first = TRUE;
61     if (first == TRUE)
62         (

```

```

63     first = FALSE;
64     pthread_mutex_init(&G_submitmutex, NULL);
65     }
66     pthread_mutex_lock(&G_submitmutex);
67 }
68

```

```

70  /*****
71  **
72  ** Routine: unlockSubmitMutex
73  **
74  ** Inputs: None
75  **
76  ** Outputs: None
77  **
78  ** Return Codes:
79  **      None
80  **
81  ** Purpose: Unlock the mutex for the submit object
82  **
83  *****/
84  */
85  static void
86  unlockSubmitMutex()
87  {
88  1  pthread_mutex_unlock(&g_submitmutex);
89  1
90  }

```

```

92  /*****
93  **
94  ** Routine: LookupsSubmitElement
95  **
96  ** Inputs:
97  **      int ID - submit object ID associated with element
98  **      int elementID - submit element ID associated with element
99  **
100  ** Outputs:
101  **      int *status - status of the function if an error occurred
102  **      EDMRESubmitElement **se - Place to put the pointer to the
103  **      element
104  **
105  ** Return Codes:
106  **      int - 0 for success or non-zero for failure
107  **
108  ** Purpose: Finds a submit element based on the submit ID and the
109  **      element ID.
110  *****/
111  */
112  int
113  RemoveSubmittrees ()
114  {
115  1  // In the future we might want to pass in the submit object
116  1  // In case they become persistent stores.
117  1
118  1  EDMRESubmitObj *so;
119  1  RMBinaryTreeIterator *submitObjectTree;
120  1
121  1  // create a binary tree iterator so that we can get each submit
122  1  // and therefore get each submit element and remove its submit file
123  1  submitObjectTree = new RMBinaryTreeIterator(submitObjects);
124  1  if (NULL == submitObjectTree)
125  1  {
126  1      return (-1);
127  1  }
128  1
129  1  //Get the iterator to start at the beginning
130  1  submitObjectTree->reset();
131  1
132  1  // while we have objects to work with we must keep getting the
133  1  // next element. This also gets the first object once a reset
134  1  // is performed like above. The () operator acts to the next
135  1  while (NULL != (*submitObjectTree) ())
136  1  {
137  1      //Get the next object out of the tree
138  1      so = (EDMRESubmitObj *)submitObjectTree->key();
139  1      so->deleteSubmitFiles();
140  1  }
141  1  //delete the submitObjects Iterator
142  1  delete submitObjectTree;
143  1  // return success
144  1  return (0);
145  1
146  }

```

```

147 int
148 LookupSubmitElement(int ID, int elementID, EDRESsubmitElement **so,
149 int *status)
150 {
151     EDRESsubmitObj *so;
152     EDRESsubmitObj *ret;
153     if (status == NULL)
154     {
155         if (status == NULL)
156         {
157             return -1;
158         }
159         *status = 0;
160         if (so == NULL)
161         {
162             *status = SUBMIT_BAD_PARAM;
163             return -1;
164         }
165         so = new EDRESsubmitObj();
166         if (so == NULL)
167         {
168             *status = SUBMIT_NO_MEMORY;
169             return -1;
170         }
171         so -> setSubmitID(ID);
172         LockSubmitMutex();
173         ret = (EDRESsubmitObj *) submitObjects.find(so);
174         delete so;
175         if (ret == NULL)
176         {
177             *status = SUBMIT_LOOKUP_FAILED;
178             unlockSubmitMutex();
179             return -1;
180         }
181         *so = ret -> getSubmitElement(elementID);
182         unlockSubmitMutex();
183         if (*so == NULL)
184         {
185             *status = SUBMIT_LOOKUP_FAILED;
186             unlockSubmitMutex();
187             return -1;
188         }
189         *so = ret;
190         return 0;
191     }
192 }

```

```

203 /*****
204 **
205 ** Routine: LookupSubmitObj
206 ** Inputs: int ID - submit object ID
207 ** Outputs: int *status - status of the function if an error occurred
208 ** Return Codes:
209 **     int - 0 for success or non-zero for failure
210 ** Purpose: Finds a submit object based on the submit ID.
211 *****/
212
213 int
214 LookupSubmitObj(int ID, EDRESsubmitObj **so, int *status)
215 {
216     EDRESsubmitObj *tmpso;
217     EDRESsubmitObj *ret;
218     if (status == NULL)
219     {
220         *status = SUBMIT_BAD_PARAM;
221         return -1;
222     }
223     if (so == NULL)
224     {
225         *status = SUBMIT_LOOKUP_FAILED;
226         return -1;
227     }
228     tmpso = new EDRESsubmitObj();
229     if (tmpso == NULL)
230     {
231         *status = SUBMIT_NO_MEMORY;
232         return -1;
233     }
234     tmpso -> setSubmitID(ID);
235     LockSubmitMutex();
236     ret = (EDRESsubmitObj *) submitObjects.find(tmpso);
237     unlockSubmitMutex();
238     delete tmpso;
239     if (ret == NULL)
240     {
241         *status = SUBMIT_LOOKUP_FAILED;
242         return -1;
243     }
244     *so = ret;
245     return 0;
246 }

```

Page 143 of 248	NewsSubmitObject	Fri Jan 04 16:35:25 2008	Page 144 of 248	NewsSubmitElement	Fri Jan 04 16:35:25 2008
<pre>255 /***** 256 ** 257 ** Routine: NewsSubmitObject 258 ** 259 ** Inputs: NONE 260 ** 261 ** Outputs: int *status - status of the function if an error occurred 262 ** 263 ** Return Codes: 264 ** int - ID of the new submit Object 265 ** 266 ** Purpose: Creates a new submit object and inserts it in the submit 267 ** tree. 268 ** 269 **/ 270 271 272 273 274 275 276 277 278 279 280 281 282 283 284 285 286 287 288 289 290 291 292 293 294 295 296 297 298 299 300 301 302 303 304 305 306 307 308 309 310 311 312 313 314 315 316 317 318 319 320 321 322 323 324 325 326 327 328 329 330 331 332 333 334 335 336 337 338 339 340 341 342 343 344 345 346 347 348 349 350 351 352 353 354 355 356 357 358 359 360 361 362 363 364 365 366 367 368 369 370 371 372 373 374 375 376 377 378 379 380 381 382 383 384 385 386 387 388 389 390 391 392 393 394 395 396 397 398 399 400 401 402 403 404 405 406 407 408 409 410 411 412 413 414 415 416 417 418 419 420 421 422 423 424 425 426 427 428 429 430 431 432 433 434 435 436 437 438 439 440 441 442 443 444 445 446 447 448 449 450 451 452 453 454 455 456 457 458 459 460 461 462 463 464 465 466 467 468 469 470 471 472 473 474 475 476 477 478 479 480 481 482 483 484 485 486 487 488 489 490 491 492 493 494 495 496 497 498 499 500 501 502 503 504 505 506 507 508 509 510 511 512 513 514 515 516 517 518 519 520 521 522 523 524 525 526 527 528 529 530 531 532 533 534 535 536 537 538 539 540 541 542 543 544 545 546 547 548 549 550 551 552 553 554 555 556 557 558 559 560 561 562 563 564 565 566 567 568 569 570 571 572 573 574 575 576 577 578 579 580 581 582 583 584 585 586 587 588 589 590 591 592 593 594 595 596 597 598 599 600 601 602 603 604 605 606 607 608 609 610 611 612 613 614 615 616 617 618 619 620 621 622 623 624 625 626 627 628 629 630 631 632 633 634 635 636 637 638 639 640 641 642 643 644 645 646 647 648 649 650 651 652 653 654 655 656 657 658 659 660 661 662 663 664 665 666 667 668 669 670 671 672 673 674 675 676 677 678 679 680 681 682 683 684 685 686 687 688 689 690 691 692 693 694 695 696 697 698 699 700 701 702 703 704 705 706 707 708 709 710 711 712 713 714 715 716 717 718 719 720 721 722 723 724 725 726 727 728 729 730 731 732 733 734 735 736 737 738 739 740 741 742 743 744 745 746 747 748 749 750 751 752 753 754 755 756 757 758 759 760 761 762 763 764 765 766 767 768 769 770 771 772 773 774 775 776 777 778 779 780 781 782 783 784 785 786 787 788 789 790 791 792 793 794 795 796 797 798 799 800 801 802 803 804 805 806 807 808 809 810 811 812 813 814 815 816 817 818 819 820 821 822 823 824 825 826 827 828 829 830 831 832 833 834 835 836 837 838 839 840 841 842 843 844 845 846 847 848 849 850 851 852 853 854 855 856 857 858 859 860 861 862 863 864 865 866 867 868 869 870 871 872 873 874 875 876 877 878 879 880 881 882 883 884 885 886 887 888 889 890 891 892 893 894 895 896 897 898 899 900 901 902 903 904 905 906 907 908 909 910 911 912 913 914 915 916 917 918 919 920 921 922 923 924 925 926 927 928 929 930 931 932 933 934 935 936 937 938 939 940 941 942 943 944 945 946 947 948 949 950 951 952 953 954 955 956 957 958 959 960 961 962 963 964 965 966 967 968 969 970 971 972 973 974 975 976 977 978 979 980 981 982 983 984 985 986 987 988 989 990 991 992 993 994 995 996 997 998 999</pre>	<pre>int NewsSubmitObject(int *status) { EDNewsSubmitObj *so; EDNewsSubmitObj *ret; if (*status == NULL) { *status = SUBMIT_NO_MEMORY; return 0; } if (*status == NULL) { return 0; } so = new EDNewsSubmitObj(); if (so == NULL) { *status = SUBMIT_NO_MEMORY; return 0; } so -> setSubmitID(++numberOfSubmitObjects); lockSubmitMutex(); ret = (EDNewsSubmitObj *) submitObjects.insert(so); unlockSubmitMutex(); if (ret == NULL) { *status = SUBMIT_INSERT_FAILED; numberOfSubmitObjects--; delete so; return 0; } return numberOfSubmitObjects; }</pre>	<pre>319 /***** 320 ** 321 ** Routine: NewsSubmitElement 322 ** 323 ** Inputs: int ID - submit ID associated with new element 324 ** int *status - status of the function if an error occurred 325 ** 326 ** Return Codes: 327 ** int - ID of the new submit element 328 ** 329 ** Purpose: Creates a new submit element and inserts it in the 330 ** element tree. 331 ** 332 **/ 333 334 335 336 337 338 339 340 341 342 343 344 345 346 347 348 349 350 351 352 353 354 355 356 357 358 359 360 361 362 363 364 365 366 367 368 369 370 371 372 373 374 375 376 377 378 379 380 381 382 383 384 385 386 387 388 389 390 391 392 393 394 395 396 397 398 399 400 401 402 403 404 405 406 407 408 409 410 411 412 413 414 415 416 417 418 419 420 421 422 423 424 425 426 427 428 429 430 431 432 433 434 435 436 437 438 439 440 441 442 443 444 445 446 447 448 449 450 451 452 453 454 455 456 457 458 459 460 461 462 463 464 465 466 467 468 469 470 471 472 473 474 475 476 477 478 479 480 481 482 483 484 485 486 487 488 489 490 491 492 493 494 495 496 497 498 499 500 501 502 503 504 505 506 507 508 509 510 511 512 513 514 515 516 517 518 519 520 521 522 523 524 525 526 527 528 529 530 531 532 533 534 535 536 537 538 539 540 541 542 543 544 545 546 547 548 549 550 551 552 553 554 555 556 557 558 559 560 561 562 563 564 565 566 567 568 569 570 571 572 573 574 575 576 577 578 579 580 581 582 583 584 585 586 587 588 589 590 591 592 593 594 595 596 597 598 599 600 601 602 603 604 605 606 607 608 609 610 611 612 613 614 615 616 617 618 619 620 621 622 623 624 625 626 627 628 629 630 631 632 633 634 635 636 637 638 639 640 641 642 643 644 645 646 647 648 649 650 651 652 653 654 655 656 657 658 659 660 661 662 663 664 665 666 667 668 669 670 671 672 673 674 675 676 677 678 679 680 681 682 683 684 685 686 687 688 689 690 691 692 693 694 695 696 697 698 699 700 701 702 703 704 705 706 707 708 709 710 711 712 713 714 715 716 717 718 719 720 721 722 723 724 725 726 727 728 729 730 731 732 733 734 735 736 737 738 739 740 741 742 743 744 745 746 747 748 749 750 751 752 753 754 755 756 757 758 759 760 761 762 763 764 765 766 767 768 769 770 771 772 773 774 775 776 777 778 779 780 781 782 783 784 785 786 787 788 789 790 791 792 793 794 795 796 797 798 799 800 801 802 803 804 805 806 807 808 809 810 811 812 813 814 815 816 817 818 819 820 821 822 823 824 825 826 827 828 829 830 831 832 833 834 835 836 837 838 839 840 841 842 843 844 845 846 847 848 849 850 851 852 853 854 855 856 857 858 859 860 861 862 863 864 865 866 867 868 869 870 871 872 873 874 875 876 877 878 879 880 881 882 883 884 885 886 887 888 889 890 891 892 893 894 895 896 897 898 899 900 901 902 903 904 905 906 907 908 909 910 911 912 913 914 915 916 917 918 919 920 921 922 923 924 925 926 927 928 929 930 931 932 933 934 935 936 937 938 939 940 941 942 943 944 945 946 947 948 949 950 951 952 953 954 955 956 957 958 959 960 961 962 963 964 965 966 967 968 969 970 971 972 973 974 975 976 977 978 979 980 981 982 983 984 985 986 987 988 989 990 991 992 993 994 995 996 997 998 999</pre>			
Page 145 of 248	./libx_restoreEDNewsSubmitAplic 7	Fri Jan 04 16:35:25 2008	Page 146 of 248	./libx_restoreEDNewsSubmitAplic 8	Fri Jan 04 16:35:25 2008

```

377  /*****
378  **
379  ** Routine: GetSEClientUserName
380  **
381  ** Inputs:      int ID - submit ID associated with element
382  **              int elementID - submit element ID associated with element
383  **              int maxsize - maximum size to fill the buffer with
384  **
385  ** Outputs:     int *status - status of the function if an error occurred
386  **              char *buff - place to put the user name
387  **
388  ** Return Codes:
389  **             int - 0 for success or non-zero for failure
390  **
391  ** Purpose:     Gets the user name of the given client.
392  **
393  *****/
394  */
395  int
396  GetSEClientUserName(int ID, int elementID, char *buff, int maxsize,
397                      int *status)
398  {
399      EPMRESsubmitElement *se;
400      int ret;
401
402      if (status == NULL)
403      {
404          return -1;
405      }
406
407      if (buff == NULL)
408      {
409          *status = SUBMIT_BAD_PARAM;
410          return -1;
411      }
412
413      ret = LookupsSubmitElement(ID, elementID, &se, status);
414
415      if (ret != 0)
416      {
417          return ret;
418      }
419
420      se -> getClientUserName(buff, maxsize);
421
422      return 0;
423  }
424

```

```

426  /*****
427  **
428  ** Routine: GetSEWorkItemName
429  **
430  ** Inputs:      int ID - submit ID associated with element
431  **              int elementID - submit element ID associated with element
432  **              int maxsize - maximum size to fill the buffer with
433  **
434  ** Outputs:     int *status - status of the function if an error occurred
435  **              char *buff - place to put the work item
436  **
437  ** Return Codes:
438  **             int - 0 for success or non-zero for failure
439  **
440  ** Purpose:     Gets the workitem name of the given submit element.
441  **
442  *****/
443  */
444  int
445  GetSEWorkItemName(int ID, int elementID, char *buff, int maxsize,
446                    int *status)
447  {
448      EPMRESsubmitElement *se;
449      int ret;
450
451      if (status == NULL)
452      {
453          return -1;
454      }
455
456      if (buff == NULL)
457      {
458          *status = SUBMIT_BAD_PARAM;
459          return -1;
460      }
461
462      ret = LookupsSubmitElement(ID, elementID, &se, status);
463
464      if (ret != 0)
465      {
466          return ret;
467      }
468
469      se -> getWorkItem(buff, maxsize);
470
471      return 0;
472  }
473

```

```

473  */
474  **
475  ** Routine: GetSETemplateName
476  **
477  ** Inputs:  int ID - submit ID associated with element
478  **          int elementID - submit element ID associated with element
479  **          int maxsize - maximum size to fill the buffer with
480  **          int *status - status of the function if an error occurred
481  **          char *buff - place to put the template
482  **
483  ** Return Codes:
484  **          int - 0 for success or non-zero for failure
485  **
486  ** Purpose: Gets the template name of the given submit element.
487  **
488  ****
489
490  */
491  int
492  GetSETemplateName(int ID, int elementID, char *buff, int maxsize,
493                  int *status)
494  {
495      EDRSubmittElement *se;
496      int ret;
497
498      if (status == NULL)
499      {
500          return -1;
501      }
502
503      if (buff == NULL)
504      {
505          *status = SUBMIT_BAD_PARAM;
506          return -1;
507      }
508
509      ret = LookupSubmittElement(ID, elementID, &se, status);
510
511      if (ret != 0)
512          return ret;
513
514      se -> getTemplate(buff, maxsize);
515
516      return 0;
517  }
518

```

```

520  */
521  **
522  ** Routine: GetSETrailSecAlternate
523  **
524  ** Inputs:  int ID - submit ID associated with element
525  **          int elementID - submit element ID associated with element
526  **          int *status - status of the function if an error occurred
527  **
528  ** Return Codes:
529  **          boolean - TRUE is yes and FALSE otherwise
530  **
531  ** Purpose: Gets the Is-Tail-Sec-Alternate boolean from submit
532  **          element.
533  **
534  ****
535
536  */
537  boolean_t
538  GetSETrailSecAlternate(int ID, int elementID,
539                      int *status)
540  {
541      EDRSubmittElement *se;
542      int ret;
543
544      if (status == NULL)
545      {
546          return FALSE;
547      }
548
549      ret = LookupSubmittElement(ID, elementID, &se, status);
550
551      if (ret != 0)
552          return FALSE;
553
554      return se -> IsAlternateTrailSec(!);
555  }
556

```

```

557  /*****
558  **
559  ** Routine: GetSESourceClientName
560  **
561  ** Inputs:   int ID - submit ID associated with element
562  **           int elementID - element ID associated with element
563  **           int maxsize - maximum size to fill the buffer with
564  **
565  ** Outputs:  int *status - status of the function if an error occurred
566  **           char *buff - place to put the client name
567  **
568  ** Return Codes:
569  **           int - 0 for success or non-zero for failure
570  **
571  ** Purpose:  Gets the source client name of the given submit element.
572  **
573  *****/
574  */
575  int
576  GetSESourceClientName(int ID, int elementID, char *buff, int maxsize,
577                        int *status)
578  {
579      EDRESsubmitElement *se;
580      int ret;
581
582      if (*status == NULL)
583      {
584          return -1;
585      }
586
587      if (buff == NULL)
588      {
589          *status = SUBMIT_BAD_PARAM;
590          return -1;
591      }
592
593      ret = LookupSubmitElement(ID, elementID, &se, status);
594
595      if (ret != 0)
596          return ret;
597
598      se -> getClientSource(buff, maxsize);
599
600      return 0;
601  }
602

```

```

603  /*****
604  **
605  ** Routine: GetSEDestClientName
606  **
607  ** Inputs:   int ID - submit ID associated with element
608  **           int elementID - submit element ID associated with element
609  **           int maxsize - maximum size to fill the buffer with
610  **
611  ** Outputs:  int *status - status of the function if an error occurred
612  **           char *buff - place to put the client name
613  **
614  ** Return Codes:
615  **           int - 0 for success or non-zero for failure
616  **
617  ** Purpose:  Gets the destination client name of the given submit
618  **           element.
619  **
620  *****/
621  */
622  int
623  GetSEDestClientName(int ID, int elementID, char *buff, int maxsize,
624                      int *status)
625  {
626      EDRESsubmitElement *se;
627      int ret;
628
629      if (*status == NULL)
630      {
631          return -1;
632      }
633
634      if (buff == NULL)
635      {
636          *status = SUBMIT_BAD_PARAM;
637          return -1;
638      }
639
640      ret = LookupSubmitElement(ID, elementID, &se, status);
641
642      if (ret != 0)
643          return ret;
644
645      se -> getClientDestName(buff, maxsize);
646
647      return 0;
648  }
649

```



```

651 /*****
652 **
653 ** Routine: GetSetIsRestoreInPlace
654 **
655 ** Inputs:   int ID - submit ID associated with element
656             int elementID - submit element ID associated with element
657 **
658 ** Outputs:  int *status - status of the function if an error occurred
659 **
660 ** Return Codes:
661             boolean_Cy - TRUE is yes and FALSE otherwise
662 **
663 ** Purpose:  Gets the Is-Restore-In-Place boolean from submit element.
664 *****/
665
666

```

```

668
669 boolean_Cy
670 GetSetIsRestoreInPlace (int ID, int elementID,
671                        int *status)
672 {
673     EDRESSubmitElement *se;
674     int ret;
675
676     if (status == NULL)
677     {
678         return FALSE;
679     }
680
681     ret = LookUpSubmitElement(ID, elementID, &se, status);
682     if (ret != 0)
683     {
684         return FALSE;
685     }
686     return se -> isInPlace();
687 }

```

```

688 /*****
689 **
690 ** Routine: GetSetDestDirTop
691 **
692 ** Inputs:   int ID - submit ID associated with element
693             int elementID - submit element ID associated with element
694             int maxsize - maximum size to fill the buffer with
695 **
696 ** Outputs:  int *status - status of the function if an error occurred
697             char *buff - place to put the destination directory
698 **
699 ** Return Codes:
700             int - 0 for success or non-zero for failure
701 **
702 ** Purpose:  Gets the destination directory of the given submit
703             element.
704 *****/
705
706

```

```

707 int
708 GetSetDestDirTop (int ID, int elementID, char *buff, int maxsize,
709                  int *status)
710 {
711     EDRESSubmitElement *se;
712     int ret;
713
714     if (status == NULL)
715     {
716         return -1;
717     }
718     if (buff == NULL)
719     {
720         return -1;
721     }
722     *status = SUBMIT_BAD_PARAM;
723     return -1;
724 }
725
726 ret = LookUpSubmitElement(ID, elementID, &se, status);
727 if (ret != 0)
728 {
729     return ret;
730 }
731 se -> getDirectoryTop(buff, maxsize);
732 return 0;
733 }

```

```

735 /*****
736 **
737 ** Routine: GetSEDestDirectory
738 **
739 ** Inputs:  int ID - submit ID associated with element
740 **          int elementID - submit element ID associated with element
741 **          int maxSize - maximum size to fill the buffer with
742 ** Outputs: int *status - status of the function if an error occurred
743 **          char *buff - place to put the directory
744 **
745 ** Return Codes:
746 **             int = 0 for success or non-zero for failure
747 **             int *status
748 **
749 ** Purpose: Gets the destination directory of the given submit
750 **          element.
751 **
752 **/
753
754 int
755 GetSEDestDirectory(int ID, int elementID, char *buff, int maxSize,
756                   int *status)
757 {
758     EIMRESSubmitElement *se;
759     int ret;
760
761     if (status == NULL)
762     {
763         return -1;
764     }
765     if (buff == NULL)
766     {
767         *status = SUBMIT_BAD_PARAM;
768         return -1;
769     }
770
771     ret = LookUpSubmitElement(ID, elementID, &se, status);
772
773     if (ret != 0)
774         return ret;
775
776     se -> getDirectoryDestination(buff, maxSize);
777
778     return 0;
779 }

```

```

762 /*****
763 **
764 ** Routine: GetSEDestOverWritePolicy
765 **
766 ** Inputs:  int ID - submit ID associated with element
767 **          int elementID - submit element ID associated with element
768 **          int *status - status of the function if an error occurred
769 **
770 ** Return Codes:
771 **             OverwritePolicy
772 **
773 ** Purpose: Gets the OverwritePolicy enum from submit element.
774 **
775 **
776 **/
777
778 OverwritePolicy
779 GetSEDestOverWritePolicy(int ID, int elementID,
780                          int *status)
781 {
782     EIMRESSubmitElement *se;
783     int ret;
784
785     if (status == NULL)
786     {
787         return Older_Only_Overwrite;
788         /* avoid warning: use default value */
789     }
790
791     ret = LookUpSubmitElement(ID, elementID, &se, status);
792
793     if (ret != 0)
794         return Older_Only_Overwrite;
795     /* avoid warning: use default value */
796
797     return se -> getOverwritePolicy();
798 }

```

```

819
820
821 **
822 ** Routine: GetSubmittFile
823
824 ** Inputs:  int ID - submit ID associated with element
825            int elementID - submit element ID associated with element
826            int maxsize - maximum size to fill the buffer with
827
828 ** Outputs:  int *status - status of the function if an error occurred
829            char *buff - place to put the submit file name
830
831 ** Return Codes:
832            int - 0 for success or non-zero for failure
833
834 ** Purpose:  Gets the submit file name of the given submit element.
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000

```

```

866
867
868 **
869 ** Routine: GetScriptName
870
871 ** Inputs:  int ID - submit ID associated with element
872            int elementID - submit element ID associated with element
873            int maxsize - maximum size to fill the buffer with
874
875 ** Outputs:  int *status - status of the function if an error occurred
876            char *buff - place to put the client script name
877
878 ** Return Codes:
879            int - 0 for success or non-zero for failure
880
881 ** Purpose:  Gets the client script name of the given submit element.
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000

```

```

913  /*
914  **
915  ** Routine: GetServerConnct
916  **
917  ** Inputs:  int ID - submtic ID associated with element
918  **          int elementID - submtic element ID associated with element
919  **          int maxSize - maximum size to fill the buffer with
920  **
921  ** Outputs: int *status - status of the function if an error occurred
922  **          char *buff - place to put the client name
923  **
924  ** Return Codes:
925  **              int - 0 for success or non-zero for failure
926  **
927  ** Purpose: Gets the client name and port to connect on for the given
928  **          submtic element.
929  **
930  **
931  ****
932  */
933
934  int
935  GetServerConnct(int ID, int elementID, char *buff, int maxSize,
936                  int *port, int *status)
937  {
938      EIMRSSubmitElement *se;
939      int ret;
940
941      if (se == NULL)
942      {
943          return -1;
944      }
945
946      if (port == NULL || buff == NULL)
947      {
948          *status = SUBMIT_BAD_PARAM;
949          return -1;
950      }
951
952      ret = LookupSubmitElement(ID, elementID, &se, status);
953
954      if (ret != 0)
955      {
956          return ret;
957      }
958
959      se -> getSocketHost(buff, maxSize);
960      *port = se -> getSocketPort();
961
962      return 0;
963  }

```

```

963  /*
964  **
965  ** Routine: GetServerMarkedSummary
966  **
967  ** Inputs:  int ID - submtic ID associated with element
968  **          int elementID - submtic element ID associated with element
969  **
970  ** Outputs: int *status - status of the function if an error occurred
971  **          struct mark_summary *buff - place to put the mark summary
972  **
973  ** Return Codes:
974  **              int - 0 for success or non-zero for failure
975  **
976  ** Purpose: Gets the mark summary of the given submtic element.
977  **
978  **
979  ****
980  */
981
982  int
983  GetServerMarkedSummary(int ID, int elementID, struct mark_summary *buff,
984                          int *status)
985  {
986      EIMRSSubmitElement *se;
987      int ret;
988
989      if (se == NULL)
990      {
991          return -1;
992      }
993
994      if (buff == NULL)
995      {
996          *status = SUBMIT_BAD_PARAM;
997          return -1;
998      }
999
1000      ret = LookupSubmitElement(ID, elementID, &se, status);
1001
1002      if (ret != 0)
1003      {
1004          return ret;
1005      }
1006
1007      se -> getMarkSummary(buff);
1008
1009      return 0;
1010  }

```

```

1009 //*****
1010 **
1011 ** Routine: GetSubVolumeNeeded
1012 **
1013 ** Inputs:  int ID - submit ID associated with element
1014 **          int elementID - submit element ID associated with element
1015 **
1016 ** Outputs: int *status - status of the function if an error occurred
1017 **          ebyl_volIdList_Cy **buff - place to put the vol list
1018 **
1019 ** Return Codes:
1020 **          int - 0 for success or non-zero for failure
1021 **
1022 ** Purpose: Gets the vol list of the given submit element.
1023 **
1024 **
1025 */
1026
1027 int
1028 GetSubVolumeNeeded(int ID, int elementID, ebyl_volIdList_Cy **buff,
1029                   int *status)
1030 {
1031     EDRESsubmitElement *se;
1032     int ret;
1033
1034     if (status == NULL)
1035     {
1036         return -1;
1037     }
1038
1039     if (buff == NULL)
1040     {
1041         *status = SUBMIT_BAD_PARAM;
1042         return -1;
1043     }
1044
1045     ret = LookupSubmitElement(ID, elementID, se, status);
1046
1047     if (ret != 0)
1048     {
1049         return ret;
1050     }
1051     se -> getVolIdList(buff);
1052     return 0;
1053 }

```

```

1055 //*****
1056 **
1057 ** Routine: GetWorkItemtype
1058 **
1059 ** Inputs:  int ID - submit ID associated with element
1060 **          int elementID - submit element ID associated with element
1061 **
1062 ** Outputs: int *status - status of the function if an error occurred
1063 **          char *type - the work item type
1064 **
1065 ** Return Codes:
1066 **          int - 0 for success or non-zero for failure
1067 **
1068 ** Purpose: Gets the vol list of the given submit element.
1069 **
1070 **
1071 */
1072
1073 int
1074 GetWorkItemtype(int ID, int elementID, char *type,
1075                int *status)
1076 {
1077     EDRESsubmitElement *se;
1078     int ret;
1079
1080     if (status == NULL)
1081     {
1082         return -1;
1083     }
1084
1085     if (type == NULL)
1086     {
1087         *status = SUBMIT_BAD_PARAM;
1088         return -1;
1089     }
1090
1091     ret = LookupSubmitElement(ID, elementID, se, status);
1092
1093     if (ret != 0)
1094     {
1095         return ret;
1096     }
1097     *type = se -> getWorkItemtype();
1098     return 0;
1099 }

```

```

1101 /*****
1102 **
1103 ** Routine: GetSEPluginData
1104 **
1105 ** Inputs:  int ID - submit ID
1106             int elementID - submit element ID
1107 **
1108 ** Outputs: int *status - a status of the operation
1109             int *plugin_data - ptr to plugin specific
1110                 submit element data
1111 **
1112 ** Return Codes:
1113             int - 0 for success and non-zero otherwise
1114 **
1115 ** Purpose: Get the pointer to the plugin specific data for the
1116             submit element
1117             *****/
1118
1119 int
1120 GetSEPluginData(
1121     int ID, int elementID, RMCcollection **plugin_data, int *status)
1122 {
1123     EDRESsubmitElement *se;
1124     int ret;
1125
1126     if (status == NULL)
1127     {
1128         return -1;
1129     }
1130     if (plugin_data == NULL)
1131     {
1132         *status = SUBMIT_BAD_PARAM;
1133         return -1;
1134     }
1135
1136     ret = LookUpSubmitElement(ID, elementID, &se, status);
1137     if (ret != 0)
1138         return ret;
1139
1140     *plugin_data = se -> getPluginData();
1141     return 0;
1142 }
1143
1144 )

```

```

1146 /*****
1147 **
1148 ** Routine: GetSOBackupAdmin
1149 **
1150 ** Inputs:  int ID - submit ID associated with element
1151             int *status - status of the function if an error occurred
1152 **
1153 ** Outputs: boolean_ty - TRUE if the user is and FALSE otherwise.
1154 **
1155 ** Return Codes:
1156             boolean_ty - TRUE if the user is and FALSE otherwise.
1157 **
1158 ** Purpose: Determine whether the user is backup admin.
1159 *****/
1160
1161 bool
1162 GetSOBackupAdmin(int ID, int *status)
1163 {
1164     EDRESsubmitObj *so;
1165     int ret;
1166
1167     if (status == NULL)
1168     {
1169         return FALSE;
1170     }
1171
1172     ret = LookUpSubmitObj(ID, &so, status);
1173     if (ret != 0)
1174         return FALSE;
1175
1176     return so -> isBackupAdmin();
1177 }
1178
1179 )

```

```

1181  /*****
1182  **
1183  ** Routine: GetSOSourcesSystemAdmin
1184  **
1185  ** Inputs:  int ID - submit ID associated with element
1186  **
1187  ** Outputs: int *status - status of the function if an error occurred
1188  **
1189  ** Return Codes:
1190  **          boolean, ly - TRUE if the user is and FALSE otherwise.
1191  **
1192  ** Purpose: Determine whether the user is source system admin.
1193  **
1194  *****/
1195  */

1197  boolean ly
1198  GetSOSourcesSystemAdmin(int ID, int *status)
1199  {
1200  { EDRESsubmitObj *so;
1201  int rec;

1202  if (status == NULL)
1203  {
1204  return FALSE;
1205  }

1206  rec = LookupSubmitObject(ID, &so, status);

1207  if (rec != 0)
1208  return FALSE;
1209  return so -> IsSourceSystemAdmin();
1210  }
1211  }
1212  }
1213  }
1214  }

```

```

1216  /*****
1217  **
1218  ** Routine: GetSODestinationAdmin
1219  **
1220  ** Inputs:  int ID - submit ID associated with element
1221  **
1222  ** Outputs: int *status - status of the function if an error occurred
1223  **
1224  ** Return Codes:
1225  **          boolean, ly - TRUE if the user is and FALSE otherwise.
1226  **
1227  ** Purpose: Determine whether the user is destination system admin.
1228  **
1229  *****/
1230  */

1232  boolean ly
1233  GetSODestinationAdmin(int ID, int *status)
1234  {
1235  { EDRESsubmitObj *so;
1236  int rec;

1237  if (status == NULL)
1238  {
1239  return FALSE;
1240  }

1241  rec = LookupSubmitObject(ID, &so, status);

1242  if (rec != 0)
1243  return FALSE;
1244  return so -> IsDestinationAdmin();
1245  }
1246  }
1247  }
1248  }
1249  }

```

```

1251 /*****
1252 **
1253 ** Routine: GetSOUserID
1254 **
1255 ** Inputs:   int ID - submit ID associated with element
1256 ** Outputs:  int *status - status of the function if an error occurred
1257 **           uid_t *userid - place to put the user ID
1258 **
1259 ** Return Codes:
1260 **           int - 0 if success and non-zero otherwise
1261 **
1262 ** Purpose:  Get the user ID.
1263 **
1264 **
1265 *****/
1266 */

```

```

1268 int
1269 GetSOUserID(int ID, uid_t *userid, int *status)
1270 {
1271     EDRESsubmitObj *so;
1272     int ret;
1273
1274     if (status == NULL)
1275     {
1276         return -1;
1277     }
1278
1279     if (userid == NULL)
1280     {
1281         *status = SUBMIT_BAD_PARAM;
1282         return -1;
1283     }
1284
1285     ret = LookupSubmitObject(ID, &so, status);
1286
1287     if (ret != 0)
1288     {
1289         return ret;
1290     }
1291
1292     *userid = so -> getUserID();
1293     return 0;
1294 }
1295

```

```

1297 /*****
1298 **
1299 ** Routine: GetSOEffectiveUID
1300 **
1301 ** Inputs:   int ID - submit ID
1302 ** Outputs:  int *status - status of the function if an error occurred
1303 **           uid_t *userid - place to put the user ID
1304 **
1305 ** Return Codes:
1306 **           int - 0 if success and non-zero otherwise
1307 **
1308 ** Purpose:  Get the effective user ID.
1309 **
1310 **
1311 *****/
1312 */

```

```

1314 int
1315 GetSOEffectiveUID(int ID, uid_t *userid, int *status)
1316 {
1317     EDRESsubmitObj *so;
1318     int ret;
1319
1320     if (status == NULL)
1321     {
1322         return -1;
1323     }
1324
1325     if (userid == NULL)
1326     {
1327         *status = SUBMIT_BAD_PARAM;
1328         return -1;
1329     }
1330
1331     ret = LookupSubmitObject(ID, &so, status);
1332
1333     if (ret != 0)
1334     {
1335         return ret;
1336     }
1337
1338     *userid = so -> getEffectiveUID();
1339     return 0;
1340 }
1341

```



```

1343  /*****
1344  **
1345  ** Routine: GetSOUserNmame
1346  **
1347  ** Inputs:   int ID - submit ID
1348  **           int maxsize - maximum size to fill the buffer with
1349  **
1350  ** Outputs:  int *status - status of the function if an error occurred
1351  **           char *buff - place to put the user name
1352  **
1353  ** Return Codes:
1354  **           int - 0 for success or non-zero for failure
1355  **
1356  ** Purpose:  Gets the user name to be used in the restore.
1357  **
1358  *****/
1359  */
1360  int
1361  GetSOUserNmame(int ID, char *buff, int maxsize, int *status)
1362  {
1363      EDRESsubmitObj *so;
1364      int ret;
1365
1366      if (status == NULL)
1367      {
1368          return -1;
1369      }
1370
1371      if (buff == NULL)
1372      {
1373          *status = SUBMIT_BAD_PARAM;
1374          return -1;
1375      }
1376
1377      ret = LookupSubmitObj(ID, &so, status);
1378
1379      if (ret != 0)
1380      {
1381          return ret;
1382      }
1383
1384      so -> getUsername(buff, maxsize);
1385
1386      return 0;
1387  }
1388

```

```

1390  /*****
1391  **
1392  ** Routine: GetSOEffectiveUserName
1393  **
1394  ** Inputs:   int ID - submit ID
1395  **           int maxsize - maximum size to fill the buffer with
1396  **
1397  ** Outputs:  int *status - status of the function if an error occurred
1398  **           char *buff - place to put the effective user name
1399  **
1400  ** Return Codes:
1401  **           int - 0 for success or non-zero for failure
1402  **
1403  ** Purpose:  Gets the effective user name to be used in the restore.
1404  **
1405  *****/
1406  */
1407  int
1408  GetSOEffectiveUserName(int ID, char *buff, int maxsize, int *status)
1409  {
1410      EDRESsubmitObj *so;
1411      int ret;
1412
1413      if (status == NULL)
1414      {
1415          return -1;
1416      }
1417
1418      if (buff == NULL)
1419      {
1420          *status = SUBMIT_BAD_PARAM;
1421          return -1;
1422      }
1423
1424      ret = LookupSubmitObj(ID, &so, status);
1425
1426      if (ret != 0)
1427      {
1428          return ret;
1429      }
1430
1431      so -> getEffectiveUserName(buff, maxsize);
1432
1433      return 0;
1434  }
1435

```

```

1433 /
1434 ****
1435
1436 3438
1439 ** routine: GetStatusSummary
1440
1441 **
1442 ** Inputs:  int ID - submit ID
1443
1444 **
1445 ** Outputs:  int *status - status of the function if an error occurred
1446             struct mark_summary *buff - place to put the mark summary
1447
1448 **
1449 ** Return Codes:
1450             int - 0 for success or non-zero for failure
1451
1452 **
1453 ** Purpose:  Get the mark summary of the given submit object.
1454
1455 ****
1456 ****
1457 ****
1458 ****
1459 ****
1460 ****
1461 ****
1462 ****
1463 ****
1464 ****
1465 ****
1466 ****
1467 ****
1468 ****
1469 ****
1470 ****
1471 ****
1472 ****
1473 ****
1474 ****
1475 ****
1476 ****
1477 ****
1478 ****
1479 ****
1480 ****
1481 ****
1482 ****
1483 ****
1484 ****
1485 ****
1486 ****
1487 ****
1488 ****
1489 ****
1490 ****
1491 ****
1492 ****
1493 ****
1494 ****
1495 ****
1496 ****
1497 ****
1498 ****
1499 ****
1500 ****
1501 ****
1502 ****
1503 ****
1504 ****
1505 ****
1506 ****
1507 ****
1508 ****
1509 ****
1510 ****
1511 ****
1512 ****
1513 ****
1514 ****
1515 ****
1516 ****
1517 ****
1518 ****
1519 ****
1520 ****
1521 ****
1522 ****
1523 ****
1524 ****
1525 ****
1526 ****
1527 ****
1528 ****
1529 ****
1530 ****
1531 ****
1532 ****
1533 ****
1534 ****
1535 ****
1536 ****
1537 ****
1538 ****
1539 ****
1540 ****
1541 ****
1542 ****
1543 ****
1544 ****
1545 ****
1546 ****
1547 ****
1548 ****
1549 ****
1550 ****
1551 ****
1552 ****
1553 ****
1554 ****
1555 ****
1556 ****
1557 ****
1558 ****
1559 ****
1560 ****
1561 ****
1562 ****
1563 ****
1564 ****
1565 ****
1566 ****
1567 ****
1568 ****
1569 ****
1570 ****
1571 ****
1572 ****
1573 ****
1574 ****
1575 ****
1576 ****
1577 ****
1578 ****
1579 ****
1580 ****
1581 ****
1582 ****
1583 ****
1584 ****
1585 ****
1586 ****
1587 ****
1588 ****
1589 ****
1590 ****
1591 ****
1592 ****
1593 ****
1594 ****
1595 ****
1596 ****
1597 ****
1598 ****
1599 ****
1600 ****
1601 ****
1602 ****
1603 ****
1604 ****
1605 ****
1606 ****
1607 ****
1608 ****
1609 ****
1610 ****
1611 ****
1612 ****
1613 ****
1614 ****
1615 ****
1616 ****
1617 ****
1618 ****
1619 ****
1620 ****
1621 ****
1622 ****
1623 ****
1624 ****
1625 ****
1626 ****
1627 ****
1628 ****
1629 ****
1630 ****
1631 ****
1632 ****
1633 ****
1634 ****
1635 ****
1636 ****
1637 ****
1638 ****
1639 ****
1640 ****
1641 ****
1642 ****
1643 ****
1644 ****
1645 ****
1646 ****
1647 ****
1648 ****
1649 ****
1650 ****
1651 ****
1652 ****
1653 ****
1654 ****
1655 ****
1656 ****
1657 ****
1658 ****
1659 ****
1660 ****
1661 ****
1662 ****
1663 ****
1664 ****
1665 ****
1666 ****
1667 ****
1668 ****
1669 ****
1670 ****
1671 ****
1672 ****
1673 ****
1674 ****
1675 ****
1676 ****
1677 ****
1678 ****
1679 ****
1680 ****
1681 ****
1682 ****
1683 ****
1684 ****
1685 ****
1686 ****
1687 ****
1688 ****
1689 ****
1690 ****
1691 ****
1692 ****
1693 ****
1694 ****
1695 ****
1696 ****
1697 ****
1698 ****
1699 ****
1700 ****
1701 ****
1702 ****
1703 ****
1704 ****
1705 ****
1706 ****
1707 ****
1708 ****
1709 ****
1710 ****
1711 ****
1712 ****
1713 ****
1714 ****
1715 ****
1716 ****
1717 ****
1718 ****
1719 ****
1720 ****
1721 ****
1722 ****
1723 ****
1724 ****
1725 ****
1726 ****
1727 ****
1728 ****
1729 ****
1730 ****
1731 ****
1732 ****
1733 ****
1734 ****
1735 ****
1736 ****
1737 ****
1738 ****
1739 ****
1740 ****
1741 ****
1742 ****
1743 ****
1744 ****
1745 ****
1746 ****
1747 ****
1748 ****
1749 ****
1750 ****
1751 ****
1752 ****
1753 ****
1754 ****
1755 ****
1756 ****
1757 ****
1758 ****
1759 ****
1760 ****
1761 ****
1762 ****
1763 ****
1764 ****
1765 ****
1766 ****
1767 ****
1768 ****
1769 ****
1770 ****
1771 ****
1772 ****
1773 ****
1774 ****
1775 ****
1776 ****
1777 ****
1778 ****
1779 ****
1780 ****
1781 ****
1782 ****
1783 ****
1784 ****
1785 ****
1786 ****
1787 ****
1788 ****
1789 ****
1790 ****
1791 ****
1792 ****
1793 ****
1794 ****
1795 ****
1796 ****
1797 ****
1798 ****
1799 ****
1800 ****
1801 ****
1802 ****
1803 ****
1804 ****
1805 ****
1806 ****
1807 ****
1808 ****
1809 ****
1810 ****
1811 ****
1812 ****
1813 ****
1814 ****
1815 ****
1816 ****
1817 ****
1818 ****
1819 ****
1820 ****
1821 ****
1822 ****
1823 ****
1824 ****
1825 ****
1826 ****
1827 ****
1828 ****
1829 ****
1830 ****
1831 ****
1832 ****
1833 ****
1834 ****
1835 ****
1836 ****
1837 ****
1838 ****
1839 ****
1840 ****
1841 ****
1842 ****
1843 ****
1844 ****
1845 ****
1846 ****
1847 ****
1848 ****
1849 ****
1850 ****
1851 ****
1852 ****
1853 ****
1854 ****
1855 ****
1856 ****
1857 ****
1858 ****
1859 ****
1860 ****
1861 ****
1862 ****
1863 ****
1864 ****
1865 ****
1866 ****
1867 ****
1868 ****
1869 ****
1870 ****
1871 ****
1872 ****
1873 ****
1874 ****
1875 ****
1876 ****
1877 ****
1878 ****
1879 ****
1880 ****
1881 ****
1882 ****
1883 ****
1884 ****
1885 ****
1886 ****
1887 ****
1888 ****
1889 ****
1890 ****
1891 ****
1892 ****
1893 ****
1894 ****
1895 ****
1896 ****
1897 ****
1898 ****
1899 ****
1900 ****
1901 ****
1902 ****
1903 ****
1904 ****
1905 ****
1906 ****
1907 ****
1908 ****
1909 ****
1910 ****
1911 ****
1912 ****
1913 ****
1914 ****
1915 ****
1916 ****
1917 ****
1918 ****
1919 ****
1920 ****
1921 ****
1922 ****
1923 ****
1924 ****
1925 ****
1926 ****
1927 ****
1928 ****
1929 ****
1930 ****
1931 ****
1932 ****
1933 ****
1934 ****
1935 ****
1936 ****
1937 ****
1938 ****
1939 ****
1940 ****
1941 ****
1942 ****
1943 ****
1944 ****
1945 ****
1946 ****
1947 ****
1948 ****
1949 ****
1950 ****
1951 ****
1952 ****
1953 ****
1954 ****
1955 ****
1956 ****
1957 ****
1958 ****
1959 ****
1960 ****
1961 ****
1962 ****
1963 ****
1964 ****
1965 ****
1966 ****
1967 ****
1968 ****
1969 ****
1970 ****
1971 ****
1972 ****
1973 ****
1974 ****
1975 ****
1976 ****
1977 ****
1978 ****
1979 ****
1980 ****
1981 ****
1982 ****
1983 ****
1984 ****
1985 ****
1986 ****
1987 ****
1988 ****
1989 ****
1990 ****
1991 ****
1992 ****
1993 ****
1994 ****
1995 ****
1996 ****
1997 ****
1998 ****
1999 ****
2000 ****

```

```

1481  /*****
1482  **
1483  ** Routine: GetSOVolJmsStandard
1484  **
1485  ** Inputs:  int ID - submit ID
1486  **          int elementID - submit element ID associated with element
1487  **
1488  ** Outputs: int *status - status of the function If an error occurred
1489  **          epyl_volidlist_t *buff - place to put the vol list
1490  **
1491  ** Return Codes:
1492  **      int - 0 for success or non-zero for failure
1493  **
1494  ** Purpose: Gets the vol list of the given submit element.
1495  *****/
1496

```

```

1454         int
1455     getMarkSummary(int ID, struct mark_summary *buff, int *status)
1456     {
1457         EDRES*mlcObj = so;
1458         int rec;
1459         if (status == NULL)
1460             return -1;
1461     }
1462     if (buff == NULL)
1463     {
1464         *status = SUBMIT_BAD_PARAM;
1465         return -1;
1466     }
1467     rec = LookupSubnObject(ID, &so, status);
1468     if (rec != 0)
1469         return rec;
1470     so -> getMarkSummary(buff);
1471     return 0;
1472 }

```

```

1499         int
1500         {
1501             GETSOVolumeNeeded (int ID, objValidList_t **buff, int *status)
1502             {
1503                 FPMESHMETOBJ *so;
1504                 int rec;
1505                 if (status == NULL)
1506                 {
1507                     return -1;
1508                 }
1509                 if (buff == NULL)
1510                 {
1511                     *status = SUBMIT_BAD_PARAM;
1512                     return -1;
1513                 }
1514                 rec = LookupSubObjSet(ID, &so, status);
1515                 if (rec != 0)
1516                     return rec;
1517                 so -> getVolumeList(buff);
1518                 return 0;
1519             }
1520         }
1521     }
1522 }

```

```

1536 /*****
1537 **
1538 ** Routine: GetSOWorkItemCount
1539 **
1540 ** Inputs:  int ID - submit ID
1541 **
1542 ** Outputs:  int *status - status of the function if an error occurred
1543             int *wcount - place to put the work item count
1544 **
1545 ** Return Codes:
1546             int - 0 if success and non-zero otherwise
1547 **
1548 ** Purpose:  Get the user ID.
1549 **
1550 *****/
1551
1552 int
1553 GetSOWorkItemCount(int ID, int *wcount, int *status)
1554 {
1555     EDWRESsubmitObj *so;
1556     int ret;
1557
1558     if (ret == NULL)
1559     {
1560         return -1;
1561     }
1562
1563     if (wcount == NULL)
1564     {
1565         *status = SUBMIT_BAD_PARAM;
1566         return -1;
1567     }
1568
1569     ret = LookupSubmitObject(ID, &so, status);
1570
1571     if (ret != 0)
1572     {
1573         return ret;
1574     }
1575
1576     *wcount = so -> getWCount();
1577
1578     return 0;
1579 }
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622

```

```

1532 /*****
1533 **
1534 ** Routine: GetSOPrePhase
1535 **
1536 ** Inputs:  int ID - submit ID
1537             int maxsize - maximum size to fill the executable buffer
1538             int *status - status of the function if an error occurred
1539             char *executable - place to put the pre-phase executable
1540             char *prephaseargs - place to put the pre-phase arguments
1541             char **prephaseenv - place to put the pre-phase environment vars
1542             int *ret - 0 for success or non-zero for failure
1543 **
1544 ** Return Codes:
1545             int - 0 for success or non-zero for failure
1546 **
1547 ** Purpose:  Gets all pertinent information regarding the pre-phase
1548             routine.
1549 *****/
1550
1551 int
1552 GetSOPrePhase(int ID, char *executable, int maxsize,
1553               char **prephaseargs, char **prephaseenv, int *status)
1554 {
1555     EDWRESsubmitObj *so;
1556     int ret;
1557
1558     if (ret == NULL)
1559     {
1560         return -1;
1561     }
1562
1563     if (executable == NULL || prephaseargs == NULL || prephaseenv ==
1564         NULL)
1565     {
1566         *status = SUBMIT_BAD_PARAM;
1567         return -1;
1568     }
1569
1570     ret = LookupSubmitObject(ID, &so, status);
1571
1572     if (ret != 0)
1573     {
1574         return ret;
1575     }
1576
1577     so -> getPrePhaseExecutable(executable, maxsize);
1578
1579     so -> getPrePhaseArgs(prephaseargs);
1580
1581     so -> getPrePhaseEnv(prephaseenv);
1582
1583     return 0;
1584 }
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622

```

```

1624 */
1625
1626 ** Routine: GetSOExecPhase
1627 **
1628 ** Inputs:
1629 **     int ID - submit ID
1630 **     int maxsize - maximum size to fill the executable buffer
1631 **     with
1632
1633 ** Outputs:
1634 **     int *status - status of the function if an error occurred
1635 **     char *executable - place to put the executable name
1636 **     executionphasesargs - place to put the
1637 **         execution-phase arguments
1638 **     char **executionphasesenv - place to put the
1639 **         execution-phase environment vars
1640
1641 ** Return Codes:
1642 **     int - 0 for success or non-zero for failure
1643
1644 ** Purpose: Gets all pertinent information regarding the
1645 **         execution-phase runtime.
1646
1647 **
1648
1649
1650
1651
1652
1653
1654
1655 int
1656 GetSOExecPhase(int ID, char *executable, int maxsize,
1657               char **executionphasesargs, char **executionphasesenv,
1658               int *status)
1659 {
1660     EDPRSSubmitObj *so;
1661     int ret;
1662
1663     if (status == NULL)
1664     {
1665         return -1;
1666     }
1667
1668     if (executable == NULL || executionphasesargs == NULL ||
1669         executionphasesenv == NULL)
1670     {
1671         *status = SUBMIT_BAD_PARAM;
1672         return -1;
1673     }
1674
1675     ret = LookUpSubmitObjact(ID, &so, status);
1676
1677     if (ret != 0)
1678         return ret;
1679
1680     so->GetExecPhaseExecutable(executable, maxsize);
1681
1682     so->GetExecPhaseArgs(executionphasesargs);
1683
1684     so->GetExecPhaseEnv(executionphasesenv);
1685
1686     return 0;
1687 }

```

```

1677 /*****
1678 **
1679 **
1680 **
1681 **
1682 **
1683 **
1684 **
1685 **
1686 **
1687 **
1688 **
1689 **
1690 **
1691 **
1692 **
1693 **
1694 **
1695 */
1696
1697 int
1698 GetPostPhase(int ID, char *executable, int maxsize,
1699             char ***postphaseargs, char ***postphaseenv, int *status)
1700 {
1701     ERMSSubmitObj *so;
1702     int ret;
1703
1704     if (status == NULL)
1705     {
1706         return -1;
1707     }
1708     if (executable == NULL || postphaseargs == NULL || postphaseenv ==
1709         NULL)
1710     {
1711         *status = SMSGITL_BAD_PARAM;
1712         return -1;
1713     }
1714     ret = LookupSubmitObj(ID, &so, status);
1715     if (ret != 0)
1716         return ret;
1717     so -> GetPostPhaseExecutable(executable, maxsize);
1718     so -> GetPostPhaseArgs(postphaseargs);
1719     so -> GetPostPhaseEnv(postphaseenv);
1720     return 0;
1721 }
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2330
2331
2332
2333
2334
2335
2336
2337
2338
2339
2340
2341
2342
2343
2344
2345
2346
2347
2348
2349
2350
2351
2352
2353
2354
2355
2356
2357
2358
2359
2360
2361
2362
2363
2364
2365
2366
2367
2368
2369
2370
2371
2372
2373
2374
2375
2376
2377
2378
2379
2380
2381
2382
2383
2384
2385
2386
2387
2388
2389
2390
2391
2392
2393
2394
2395
2396
2397
2398
2399
2400
2401
2402
2403
2404
2405
2406
2407
2408
2409
2410
2411
2412
2413
2414
2415
2416
2417
2418
2419
2420
2421
2422
2423
2424
2425
2426
2427
2428
2429
2430
2431
2432
2433
2434
2435
2436
2437
2438
2439
2440
2441
2442
2443
2444
2445
2446
2447
2448
2449
2450
2451
2452
2453
2454
245
```

```

1729 /*****
1730 **
1731 ** Routine: IsValidSubmitID
1732 ** Inputs:   int ID - submit ID
1733 **
1734 ** Outputs:  None
1735 **
1736 ** Return Codes:
1737 **           boolean_Cy - TRUE a good ID and FALSE otherwise.
1738 **
1739 ** Purpose:  Returns whether the submit ID is good or not.
1740 **
1741 **
1742 *****/
1743 */
1744
1745 boolean_Cy
1746 IsValidSubmitID(int ID)
1747 {
1748     EDMSSubmitObj *so;
1749     int ret, status;
1750
1751     if (ID == 0)
1752         return FALSE;
1753
1754     ret = LookupSubmitObject(ID, &so, &status);
1755
1756     if (ret != 0)
1757     {
1758         return FALSE;
1759     }
1760
1761     return TRUE;
1762 }

```

```

1764 /*****
1765 **
1766 ** Routine: SetSOBasics
1767 ** Inputs:   int ID - submit ID
1768 **           unsigned int type - restore type
1769 **           unsigned int flags - execution flags
1770 **
1771 ** Outputs:  int *status - a status of the operation
1772 **
1773 ** Return Codes:
1774 **           int - 0 for success and non-zero otherwise
1775 **
1776 ** Purpose:  Sets some submit object internals.
1777 **
1778 *****/
1779 */
1780
1781 int
1782 SetSOBasics(int ID, unsigned int type, unsigned int flags,
1783             int *status)
1784 {
1785     EDMSSubmitObj *so;
1786     int ret;
1787
1788     if (status == NULL)
1789     {
1790         return -1;
1791     }
1792
1793     ret = LookupSubmitObject(ID, &so, status);
1794
1795     if (ret != 0)
1796     {
1797         return ret;
1798     }
1799
1800     so -> setSubmitType(type);
1801     so -> setExecutionFlags(flags);
1802
1803     return 0;
1804 }
1805

```

```

1807 /*****
1808 **
1809 ** Routine: SetSovMCheck
1810 **
1811 ** Inputs:  int ID, submit ID
1812 **          int ID, program type
1813 **          unsigned int flags - execution flags
1814 **
1815 ** Outputs: int *status - a status of the operation
1816 **
1817 ** Return Codes:
1818 **              int - 0 for success and non-zero otherwise
1819 **
1820 ** Purpose: Sets vm check variable.
1821 ****
1822 ****
1823 */
1824
1825 int
1826 SetSovMCheck(int ID, bool&by_vncheck, int *status)
1827 {
1828     EDRESsubmitObj *so;
1829     int ret;
1830
1831     if (status == NULL)
1832     {
1833         return -1;
1834     }
1835
1836     ret = LookUpSubmitObj(ID, &so, status);
1837
1838     if (ret != 0)
1839     {
1840         return ret;
1841     }
1842
1843     so -> setSovMCheck(vncheck);
1844
1845     return 0;
1846 }

```

```

1848 /*****
1849 **
1850 ** Routine: SetSOPrePhase
1851 **
1852 ** Inputs:  int ID, submit ID
1853 **          char *executable - executable to run in the pre-phase
1854 **          char *env - environment to use in the pre-phase
1855 **
1856 ** Outputs: int *status - a status of the operation
1857 **
1858 ** Return Codes:
1859 **              int - 0 for success and non-zero otherwise
1860 **
1861 ** Purpose: Sets pre-phase variables.
1862 ****
1863 ****
1864 ****
1865 */
1866
1867 int
1868 SetSOPrePhase(int ID, char *executable, char **args, char **env,
1869               int *status)
1870 {
1871     EDRESsubmitObj *so;
1872     int ret;
1873
1874     if (status == NULL)
1875     {
1876         return -1;
1877     }
1878
1879     if (executable == NULL)
1880     {
1881         *status = SUBMIT_BAD_PARAM;
1882         return -1;
1883     }
1884
1885     ret = LookUpSubmitObj(ID, &so, status);
1886
1887     if (ret != 0)
1888     {
1889         return ret;
1890     }
1891
1892     so -> setPrePhaseExecutable(executable);
1893
1894     if (args != NULL)
1895     {
1896         so -> setPrePhaseArgs(args);
1897     }
1898
1899     if (env != NULL)
1900     {
1901         so -> setPrePhaseEnv(env);
1902     }
1903
1904     return 0;
1905 }

```

```

1303  /*****
1304  **
1305  ** Routine: SetSOExecutePhase
1306  **
1307  ** Inputs:  int ID - submit ID
1308  **          char *executable - executable to run in the execute-phase
1309  **          char *args - args to use in the execute-phase
1310  **          char *env - environment to use in the execute-phase
1311  **
1312  ** Outputs: int *status - a status of the operation
1313  **
1314  ** Return Codes:
1315  **             int - 0 for success and non-zero otherwise
1316  **
1317  ** Purpose: Sets execute-phase variables.
1318  **
1319  *****/
1320  */
1321  int
1322  SetSOExecutePhase(int ID, char *executable, char **args, char **env,
1323                   int *status)
1324  {
1325      EDRESsubmitObj *so;
1326      int ret;
1327
1328      if (status == NULL)
1329      {
1330          return -1;
1331      }
1332
1333      if (executable == NULL && args == NULL && env == NULL)
1334      {
1335          *status = SUBMITT_BAD_PARAM;
1336          return -1;
1337      }
1338
1339      ret = LookupSubmitObject(ID, &so, status);
1340
1341      if (ret != 0)
1342      {
1343          return ret;
1344      }
1345
1346      if (executable != NULL)
1347      {
1348          so->setExecutePhaseExecutable(executable);
1349      }
1350      if (args != NULL)
1351      {
1352          so->setExecutePhaseArgs(args);
1353      }
1354      if (env != NULL)
1355      {
1356          so->setExecutePhaseEnv(env);
1357      }
1358      return 0;
1359  }

```

```

1359  /*****
1360  **
1361  ** Routine: SetSOPostPhase
1362  **
1363  ** Inputs:  int ID - submit ID
1364  **          char *executable - executable to run in the post-phase
1365  **          char *args - args to use in the post-phase
1366  **          char *env - environment to use in the post-phase
1367  **
1368  ** Outputs: int *status - a status of the operation
1369  **
1370  ** Return Codes:
1371  **             int - 0 for success and non-zero otherwise
1372  **
1373  ** Purpose: Sets post-phase variables.
1374  **
1375  *****/
1376  */
1377  int
1378  SetSOPostPhase(int ID, char *executable, char **args, char **env,
1379                int *status)
1380  {
1381      EDRESsubmitObj *so;
1382      int ret;
1383
1384      if (status == NULL)
1385      {
1386          return -1;
1387      }
1388
1389      if (executable == NULL)
1390      {
1391          *status = SUBMITT_BAD_PARAM;
1392          return -1;
1393      }
1394
1395      ret = LookupSubmitObject(ID, &so, status);
1396
1397      if (ret != 0)
1398      {
1399          return ret;
1400      }
1401
1402      if (executable != NULL)
1403      {
1404          so->setPostPhaseExecutable(executable);
1405      }
1406      if (args != NULL)
1407      {
1408          so->setPostPhaseArgs(args);
1409      }
1410      if (env != NULL)
1411      {
1412          so->setPostPhaseEnv(env);
1413      }
1414      return 0;
1415  }

```

```

2014 /*****
2015 **
2016 ** Routine: SetSOAdminID
2017 **
2018 ** Inputs:   int ID - submic ID
2019 **          boolean, cy isbackcupadmin - is the user backing admin
2020 **          boolean, cy isbackcupadmin - is the user backing admin
2021 **          boolean, cy isdestadmin - is the admin of the destination
2022 **          client
2023 **
2024 ** Outputs:  int *status - a status of the operation
2025 **
2026 ** Return Codes:
2027 **             int - 0 for success and non-zero otherwise
2028 **
2029 ** Purpose:   Sets all admin booleans.
2030 ****
2031 */
2032
2033 int
2034 SetSOAdminID(int ID, boolean, cy isbackcupadmin, boolean, cy isaccdadmin,
2035              boolean, cy isdestadmin, int *status)
2036 {
2037     EDRESultObj *so;
2038     int ret;
2039
2040     if (status == NULL)
2041     {
2042         return -1;
2043     }
2044
2045     ret = LookupSubmicObj(ID, &so, status);
2046
2047     if (ret != 0)
2048     {
2049         return ret;
2050     }
2051
2052     so -> getisbackcupadmin(isbackcupadmin);
2053     so -> getisaccdadmin(isaccdadmin);
2054     so -> setisdestadmin(isdestadmin);
2055
2056     return 0;
2057 }

```

```

2059 /*****
2060 **
2061 ** Routine: SetSOUserID
2062 **
2063 ** Inputs:   int ID - submic ID
2064 **          uid_t userid - is the user ID of the restore user
2065 **          uid_t uid - is the effective user ID of the restore user
2066 **          char *username - is the effective user name of the
2067 **                      restore user
2068 **
2069 ** Outputs:  int *status - a status of the operation
2070 **
2071 ** Return Codes:
2072 **             int - 0 for success and non-zero otherwise
2073 **
2074 ** Purpose:   Sets all user ID variables.
2075 ****
2076 */
2077
2078 int
2079 SetSOUserID(int ID, uid_t userid, char *username, uid_t uid,
2080             char *username, int *status)
2081 {
2082     EDRESultObj *so;
2083     int ret;
2084
2085     if (status == NULL)
2086     {
2087         return -1;
2088     }
2089
2090     if (userid == NULL || username == NULL)
2091     {
2092         *status = SUBMIT_BAD_PARAM;
2093         return -1;
2094     }
2095
2096     ret = LookupSubmicObj(ID, &so, status);
2097
2098     if (ret != 0)
2099     {
2100         return ret;
2101     }
2102
2103     so -> getuserid(userid);
2104     so -> seteffectiveuserid(uid);
2105     so -> setUsername(username);
2106     so -> seteffectiveusername(username);
2107
2108     return 0;
2109 }
2110

```



```

2112  /.....
2113  **
2114  ** Routine: SetSOTotalSize
2115  **
2116  ** Inputs:   int ID - submic ID
2117  **           struct mark_summary *markptr - the sum of all marks
2118  **
2119  ** Outputs:  int *status - a status of the operation
2120  **
2121  ** Return Codes:
2122  **           int - 0 for success and non-zero otherwise
2123  **
2124  ** Purpose:  Sets the internal pointer to the sum of the mark pointers
2125  **
2126  **
2127  */

```

```

2128  int
2129  SetSOTotalSize(int ID, struct mark_summary *markptr, int *status)
2130  {
2131  {
2132  EMDRESumitObj *so;
2133  int ret;
2134
2135  if (status == NULL)
2136  {
2137  return -1;
2138  }
2139
2140  if (markptr == NULL)
2141  {
2142  *status = SUBMIT_BAD_PARAM;
2143  return -1;
2144  }
2145
2146  ret = LookupSubitObj(ID, &so, status);
2147
2148  if (ret != 0)
2149  {
2150  return ret;
2151  }
2152
2153  so -> setMarkSummary(markptr);
2154
2155  return 0;
2156  }

```

```

2158  /.....
2159  **
2160  ** Routine: SetSOTotalVolumes
2161  **
2162  ** Inputs:   int ID - submic ID
2163  **           ebvl_volidlist_t *volptr - the sum of all volumes
2164  **
2165  ** Outputs:  int *status - a status of the operation
2166  **
2167  ** Return Codes:
2168  **           int - 0 for success and non-zero otherwise
2169  **
2170  ** Purpose:  Sets all admin booleans.
2171  **
2172  **
2173  */

```

```

2174  int
2175  SetSOTotalVolumes(int ID, ebvl_volidlist_t *volptr, int *status)
2176  {
2177  {
2178  EMDRESumitObj *so;
2179  int ret;
2180
2181  if (status == NULL)
2182  {
2183  return -1;
2184  }
2185
2186  if (volptr == NULL)
2187  {
2188  *status = SUBMIT_BAD_PARAM;
2189  return -1;
2190  }
2191
2192  ret = LookupSubitObj(ID, &so, status);
2193
2194  if (ret != 0)
2195  {
2196  return ret;
2197  }
2198
2199  so -> setVolumeList(volptr);
2200
2201  return 0;
2202  }

```

```

2204 /
2205 **
2206 ** Routine: SetSBasics
2207 **
2208 ** Inputs:
2209 **   int ID - submit ID
2210 **   elementID - submit element ID
2211 **   char *templateName - the template of the submit element
2212 **   boolean, ly isrestore - whether the template of the submit element
2213 **   char *clienthost - the source host of the restore for the
2214 **   submit element
2215 **   char type - the type of the work item (
2216 **       KICKER, LISTENER, ...)
2217 **
2218 ** Outputs:
2219 **   int *status - a status of the operation
2220 **
2221 ** Return Codes:
2222 **   int - 0 for success and non-zero otherwise
2223 **
2224 ** Purpose: Sets all submit element basics.
2225 **
2226 **
2227 */
2228
2229 int
2230 SetSBasics(int ID, int elementID, char *workitem,
2231            char *templateName, boolean, ly isrestore,
2232            char *clienthost, char type, int *status)
2233 {
2234     EDNSubmitElement *se;
2235     int ret;
2236
2237     if (status == NULL)
2238     {
2239         return -1;
2240     }
2241
2242     if (workitem == NULL || templateName == NULL || clienthost ==
2243         NULL)
2244     {
2245         *status = SUBMIT_BAD_PARAM;
2246         return -1;
2247     }
2248
2249     ret = LookupSubmitElement(ID, elementID, &se, status);
2250
2251     if (ret != 0)
2252     {
2253         return ret;
2254     }
2255
2256     se -> setWorkItem(workitem);
2257     se -> setTemplate(templateName);
2258     se -> setLyIsrestore(ly isrestore);
2259     se -> setClientSource(clienthost);
2260     se -> setWorkItemType(type);
2261
2262     return 0;
2263 }

```

```

2204 /
2205 **
2206 ** Routine: SetSEDestination
2207 **
2208 ** Inputs:
2209 **   int ID - submit ID
2210 **   int elementID - submit element ID
2211 **   char *clientname - the destination client of the submit
2212 **   boolean, ly inplace - whether the restore is in-place or not
2213 **   char *directory - the directory to restore to
2214 **   OverwritePolicy policy - the policy of this restore
2215 **   RestoreTransport transport - the transport of this restore
2216 **
2217 ** Outputs:
2218 **   int *status - a status of the operation
2219 **
2220 ** Return Codes:
2221 **   int - 0 for success and non-zero otherwise
2222 **
2223 ** Purpose: Sets all submit element destination variables.
2224 **
2225 **
2226 */
2227
2228 int
2229 SetSEDestination(int ID, int elementID, char *clientname,
2230                  boolean, ly inplace, char *directory,
2231                  OverwritePolicy policy, RestoreTransport transport,
2232                  int *status)
2233 {
2234     EDNSubmitElement *se;
2235     int ret;
2236
2237     if (status == NULL)
2238     {
2239         return -1;
2240     }
2241
2242     if (clientname == NULL || directory == NULL)
2243     {
2244         *status = SUBMIT_BAD_PARAM;
2245         return -1;
2246     }
2247
2248     ret = LookupSubmitElement(ID, elementID, &se, status);
2249
2250     if (ret != 0)
2251     {
2252         return ret;
2253     }
2254
2255     se -> setClientHostName(clientname);
2256     se -> setInplace(inplace);
2257     se -> setDirectoryPolicy(directory);
2258     se -> setOverwritePolicy(policy);
2259     se -> setRestoreTransport(transport);
2260
2261     return 0;
2262 }

```

```

2320 /.....
2321 **
2322 ** Routine: SetSEDirTop
2323 **
2324 ** Inputs:
2325     int ID - submit ID
2326     int elementID - submit element ID
2327     char *dirTop - the top of the directory tree for the
2328                     submit element
2329 **
2330 ** Outputs: int *status - a status of the operation
2331 **
2332 ** Return Codes:
2333     int - 0 for success and non-zero otherwise
2334 **
2335 ** Purpose: Sets the top of the directory tree for the submit element.
2336 .....
```

```

2336 */
2337
2338 int
2339 SetSEDirTop(int ID, int elementID, char *dirTop, int *status)
2340 {
2341     EDKRESSubmitElement *se;
2342     int ret;
2343
2344     if (status == NULL)
2345     {
2346         return -1;
2347     }
2348
2349     if (dirTop == NULL)
2350     {
2351         *status = SUBMIT_BAD_PARAM;
2352         return -1;
2353     }
2354
2355     ret = LookupSubmitElement(ID, elementID, &se, status);
2356
2357     if (ret != 0)
2358     {
2359         return ret;
2360     }
2361
2362     se -> setDirectoryTop(dirTop);
2363
2364     return 0;
2365 }
```

```

2367 /.....
2368 **
2369 ** Routine: SetScriptName
2370 **
2371 ** Inputs:
2372     int ID - submit ID
2373     int elementID - submit element ID
2374     char *scriptname - the script name to run on the client
2375     char *username - the user name to use when running the
2376                     script
2377 **
2378 ** Outputs: int *status - a status of the operation
2379 **
2380 ** Return Codes:
2381     int - 0 for success and non-zero otherwise
2382 **
2383 ** Purpose: Sets the script name and username to run on the client.
2384 .....
```

```

2384 */
2385
2386 int
2387 SetScriptName(int ID, int elementID, char *scriptname,
2388               char *username, int *status)
2389 {
2390     EDKRESSubmitElement *se;
2391     int ret;
2392
2393     if (status == NULL)
2394     {
2395         return -1;
2396     }
2397
2398     if (scriptname == NULL || username == NULL)
2399     {
2400         *status = SUBMIT_BAD_PARAM;
2401         return -1;
2402     }
2403
2404     ret = LookupSubmitElement(ID, elementID, &se, status);
2405
2406     if (ret != 0)
2407     {
2408         return ret;
2409     }
2410
2411     se -> setScriptName(scriptname);
2412     se -> setClientUsername(username);
2413
2414     return 0;
2415 }
```

```

2417  /*****
2418  **
2419  ** Routine: SetSEBIOConnect
2420  **
2421  ** Inputs:  int ID - submit ID
2422  **          int elementID - submit element ID
2423  **          char *hostname - give hostname to connect to
2424  **          int sockport - the port to connect to on the above host
2425  **
2426  ** Outputs: int *status - a status of the operation
2427  **
2428  ** Return Codes:
2429  **             int - 0 for success and non-zero otherwise
2430  **
2431  ** Purpose: Sets all ebci information.
2432  **
2433  *****/
2434  */
2435  int
2436  SetSEBIOConnect(int ID, int elementID, char *hostname,
2437                 int sockport, int *status)
2438  {
2439      EBMRESSubmitElement *se;
2440      int ret;
2441
2442      if (status == NULL)
2443      {
2444          return -1;
2445      }
2446
2447      if (hostname == NULL)
2448      {
2449          *status = SUBMIT_BAD_PARAM;
2450          return -1;
2451      }
2452
2453      ret = LookUpSubmitElement(ID, elementID, &se, status);
2454
2455      if (ret != 0)
2456      {
2457          return ret;
2458      }
2459
2460      se -> setSocketHost(hostname);
2461      se -> setSocketPort(sockport);
2462
2463      return 0;
2464  }

```

```

2467  /*****
2468  **
2469  ** Routine: SetSESummary
2470  **
2471  ** Inputs:  int ID - submit ID
2472  **          int elementID - submit element ID
2473  **          struct mark_summary *markptr - the marks for a given work
2474  **          item
2475  **
2476  ** Outputs: int *status - a status of the operation
2477  **
2478  ** Return Codes:
2479  **             int - 0 for success and non-zero otherwise
2480  **
2481  ** Purpose: Sets all submit element marks.
2482  **
2483  *****/
2484  */
2485  int
2486  SetSESummary(int ID, int elementID, struct mark_summary *markptr,
2487              int *status)
2488  {
2489      EBMRESSubmitElement *se;
2490      int ret;
2491
2492      if (status == NULL)
2493      {
2494          return -1;
2495      }
2496
2497      if (markptr == NULL)
2498      {
2499          *status = SUBMIT_BAD_PARAM;
2500          return -1;
2501      }
2502
2503      ret = LookUpSubmitElement(ID, elementID, &se, status);
2504
2505      if (ret != 0)
2506      {
2507          return ret;
2508      }
2509
2510      se -> setMarkSummary(markptr);
2511
2512      return 0;
2513  }

```



```

2610  /*****
2611  **
2612  ** Routine: SetSEPluginData
2613  **
2614  ** Inputs:  int ID - submic ID
2615  **           int elementID - submic element ID
2616  **           RMCcollection *plugin_data - pointer to plugin specific
2617  **           submic element data
2618  **
2619  ** Outputs: int *status - a status of the operation
2620  **
2621  ** Return Codes:
2622  **             int - 0 for success and non-zero otherwise
2623  **
2624  ** Purpose: Sets the pointer to the plugin specific data for the
2625  **           submic element
2626  **
2627  **
2628  int SetSEPluginData(
2629  {
2630  int ID, int elementID, RMCcollection *plugin_data, int *status)
2631  {
2632  int ret;
2633
2634  if (status == NULL)
2635  {
2636  return -1;
2637  }
2638
2639  if (plugin_data == NULL)
2640  {
2641  *status = SUBMIT_BAD_PARAM;
2642  return -1;
2643  }
2644
2645  ret = LookupSubmicElement(ID, elementID, &se, status);
2646
2647  if (ret != 0)
2648  {
2649  return ret;
2650  }
2651
2652  se -> setPluginData(plugin_data);
2653
2654  return 0;
2655  }

```



```

129 1      if (se_dirtop != NULL)
130 1          free(se_dirtop);
131 1
132 1      if (se_directory != NULL)
133 1          free(se_directory);
134 1
135 1      if (se_client_scriptname != NULL)
136 1          free(se_client_scriptname);
137 1
138 1      if (se_socket_host != NULL)
139 1          free(se_socket_host);
140 1
141 1      if (se_submitt_file_name != NULL)
142 1          free(se_submitt_file_name);
143 1
144 1      // Need to add some free routines for the mark summary and
145 1      // the client scriptname.
146 1      // We also have to make sure the submitt file is unlinked before
147 1      // the submitt file is freed
148 1
149 1      }
150 1
151 1      /*****
152 1      ** Routine: compareTo
153 1      **
154 1      ** Inputs:  RMCcollectable *c - a pointer to the base class type which
155 1      **           you can then cast and compare.
156 1      **
157 1      ** Outputs: None
158 1      **
159 1      ** Return Codes:
160 1      **      int - returns numbers like gsort compare (-1, 0, 1)
161 1      **
162 1      ** Purpose: Compare using the submitt element ID.
163 1      *****/
164 1
165 1      */
166 1
167 1      int
168 1      EDMRESsubmitElement::compareTo(IN const RMCcollectable *c) const
169 1      {
170 1          EDMRESsubmitElement *localcmd = (EDMRESsubmitElement *) c;
171 1
172 1          if (localcmd == NULL)
173 1              return -1;
174 1
175 1          if (localcmd -> submittElementID > submittElementID)
176 1              return 1;
177 1          else if (localcmd -> submittElementID < submittElementID)
178 1              return -1;
179 1          return 0;
180 1      }
181 1
182 1      /*****/
183 1
184 1      ** Routine: isEqual
185 1
186 1      ** Inputs:  RMCcollectable *c - a pointer to the base class type which
187 1      **           you can then cast and compare.
188 1      **
189 1      ** Outputs: None
190 1
191 1      *****/
192 1
193 1      /*****/
194 1
195 1      /*****/
196 1
197 1
198 1
199 1

```

```

199 1      return CODE_SUCCESS;
200 1
201 1      ** Routine: hash
202 1
203 1      ** Inputs:  RMCcollectable *c - a pointer to the base class type which
204 1      **           you can then cast and compare.
205 1      **
206 1      ** Outputs: None
207 1      **
208 1      ** Return Codes:
209 1      **      int - returns numbers like gsort compare (-1, 0, 1)
210 1      **
211 1      ** Purpose: Returns unique value, in this case submitt element ID.
212 1      *****/
213 1
214 1      */
215 1
216 1      unsigned
217 1      EDMRESsubmitElement::hash() const
218 1      {
219 1          return (unsigned) submittElementID;
220 1      }
221 1
222 1      /*****/
223 1
224 1      ** Routine: saveData
225 1
226 1      ** Inputs:  RMCFile *f - file pointer where data will be saved.
227 1      **
228 1      ** Outputs: None
229 1      **
230 1      ** Return Codes:
231 1      **      int - returns numbers like gsort compare (-1, 0, 1)
232 1      **
233 1      ** Purpose: Save class internal data to a file.
234 1      *****/
235 1
236 1      /*****/
237 1
238 1      /*****/
239 1
240 1
241 1
242 1
243 1
244 1
245 1
246 1
247 1
248 1
249 1
250 1
251 1
252 1
253 1
254 1
255 1
256 1
257 1
258 1
259 1
260 1
261 1
262 1
263 1
264 1
265 1
266 1
267 1
268 1
269 1
270 1
271 1
272 1
273 1
274 1
275 1
276 1
277 1
278 1
279 1
280 1
281 1
282 1
283 1
284 1
285 1
286 1
287 1
288 1
289 1
290 1
291 1
292 1
293 1
294 1
295 1
296 1
297 1
298 1
299 1
300 1
301 1
302 1
303 1
304 1
305 1
306 1
307 1
308 1
309 1
310 1
311 1
312 1
313 1
314 1
315 1
316 1
317 1
318 1
319 1
320 1
321 1
322 1
323 1
324 1
325 1
326 1
327 1
328 1
329 1
330 1
331 1
332 1
333 1
334 1
335 1
336 1
337 1
338 1
339 1
340 1
341 1
342 1
343 1
344 1
345 1
346 1
347 1
348 1
349 1
350 1
351 1
352 1
353 1
354 1
355 1
356 1
357 1
358 1
359 1
360 1
361 1
362 1
363 1
364 1
365 1
366 1
367 1
368 1
369 1
370 1
371 1
372 1
373 1
374 1
375 1
376 1
377 1
378 1
379 1
380 1
381 1
382 1
383 1
384 1
385 1
386 1
387 1
388 1
389 1
390 1
391 1
392 1
393 1
394 1
395 1
396 1
397 1
398 1
399 1
400 1
401 1
402 1
403 1
404 1
405 1
406 1
407 1
408 1
409 1
410 1
411 1
412 1
413 1
414 1
415 1
416 1
417 1
418 1
419 1
420 1
421 1
422 1
423 1
424 1
425 1
426 1
427 1
428 1
429 1
430 1
431 1
432 1
433 1
434 1
435 1
436 1
437 1
438 1
439 1
440 1
441 1
442 1
443 1
444 1
445 1
446 1
447 1
448 1
449 1
450 1
451 1
452 1
453 1
454 1
455 1
456 1
457 1
458 1
459 1
460 1
461 1
462 1
463 1
464 1
465 1
466 1
467 1
468 1
469 1
470 1
471 1
472 1
473 1
474 1
475 1
476 1
477 1
478 1
479 1
480 1
481 1
482 1
483 1
484 1
485 1
486 1
487 1
488 1
489 1
490 1
491 1
492 1
493 1
494 1
495 1
496 1
497 1
498 1
499 1
500 1
501 1
502 1
503 1
504 1
505 1
506 1
507 1
508 1
509 1
510 1
511 1
512 1
513 1
514 1
515 1
516 1
517 1
518 1
519 1
520 1
521 1
522 1
523 1
524 1
525 1
526 1
527 1
528 1
529 1
530 1
531 1
532 1
533 1
534 1
535 1
536 1
537 1
538 1
539 1
540 1
541 1
542 1
543 1
544 1
545 1
546 1
547 1
548 1
549 1
550 1
551 1
552 1
553 1
554 1
555 1
556 1
557 1
558 1
559 1
560 1
561 1
562 1
563 1
564 1
565 1
566 1
567 1
568 1
569 1
570 1
571 1
572 1
573 1
574 1
575 1
576 1
577 1
578 1
579 1
580 1
581 1
582 1
583 1
584 1
585 1
586 1
587 1
588 1
589 1
590 1
591 1
592 1
593 1
594 1
595 1
596 1
597 1
598 1
599 1
600 1
601 1
602 1
603 1
604 1
605 1
606 1
607 1
608 1
609 1
610 1
611 1
612 1
613 1
614 1
615 1
616 1
617 1
618 1
619 1
620 1
621 1
622 1
623 1
624 1
625 1
626 1
627 1
628 1
629 1
630 1
631 1
632 1
633 1
634 1
635 1
636 1
637 1
638 1
639 1
640 1
641 1
642 1
643 1
644 1
645 1
646 1
647 1
648 1
649 1
650 1
651 1
652 1
653 1
654 1
655 1
656 1
657 1
658 1
659 1
660 1
661 1
662 1
663 1
664 1
665 1
666 1
667 1
668 1
669 1
670 1
671 1
672 1
673 1
674 1
675 1
676 1
677 1
678 1
679 1
680 1
681 1
682 1
683 1
684 1
685 1
686 1
687 1
688 1
689 1
690 1
691 1
692 1
693 1
694 1
695 1
696 1
697 1
698 1
699 1
700 1
701 1
702 1
703 1
704 1
705 1
706 1
707 1
708 1
709 1
710 1
711 1
712 1
713 1
714 1
715 1
716 1
717 1
718 1
719 1
720 1
721 1
722 1
723 1
724 1
725 1
726 1
727 1
728 1
729 1
730 1
731 1
732 1
733 1
734 1
735 1
736 1
737 1
738 1
739 1
740 1
741 1
742 1
743 1
744 1
745 1
746 1
747 1
748 1
749 1
750 1
751 1
752 1
753 1
754 1
755 1
756 1
757 1
758 1
759 1
760 1
761 1
762 1
763 1
764 1
765 1
766 1
767 1
768 1
769 1
770 1
771 1
772 1
773 1
774 1
775 1
776 1
777 1
778 1
779 1
780 1
781 1
782 1
783 1
784 1
785 1
786 1
787 1
788 1
789 1
790 1
791 1
792 1
793 1
794 1
795 1
796 1
797 1
798 1
799 1
800 1
801 1
802 1
803 1
804 1
805 1
806 1
807 1
808 1
809 1
810 1
811 1
812 1
813 1
814 1
815 1
816 1
817 1
818 1
819 1
820 1
821 1
822 1
823 1
824 1
825 1
826 1
827 1
828 1
829 1
830 1
831 1
832 1
833 1
834 1
835 1
836 1
837 1
838 1
839 1
840 1
841 1
842 1
843 1
844 1
845 1
846 1
847 1
848 1
849 1
850 1
851 1
852 1
853 1
854 1
855 1
856 1
857 1
858 1
859 1
860 1
861 1
862 1
863 1
864 1
865 1
866 1
867 1
868 1
869 1
870 1
871 1
872 1
873 1
874 1
875 1
876 1
877 1
878 1
879 1
880 1
881 1
882 1
883 1
884 1
885 1
886 1
887 1
888 1
889 1
890 1
891 1
892 1
893 1
894 1
895 1
896 1
897 1
898 1
899 1
900 1
901 1
902 1
903 1
904 1
905 1
906 1
907 1
908 1
909 1
910 1
911 1
912 1
913 1
914 1
915 1
916 1
917 1
918 1
919 1
920 1
921 1
922 1
923 1
924 1
925 1
926 1
927 1
928 1
929 1
930 1
931 1
932 1
933 1
934 1
935 1
936 1
937 1
938 1
939 1
940 1
941 1
942 1
943 1
944 1
945 1
946 1
947 1
948 1
949 1
950 1
951 1
952 1
953 1
954 1
955 1
956 1
957 1
958 1
959 1
960 1
961 1
962 1
963 1
964 1
965 1
966 1
967 1
968 1
969 1
970 1
971 1
972 1
973 1
974 1
975 1
976 1
977 1
978 1
979 1
980 1
981 1
982 1
983 1
984 1
985 1
986 1
987 1
988 1
989 1
990 1
991 1
992 1
993 1
994 1
995 1
996 1
997 1
998 1
999 1
1000 1

```

```

253 EDMResSubItemElement::saveOutputs(IN RWFile &f)
254 {
255     // Save parent class data too
256     RWCollectable::saveOutputs(f);
257
258     // Left as an exercise
259 }
260
261 /*****
262
263 ** Routine: saveOutputs
264
265 ** Inputs:  RWOutputStream strm - stream to write internal data to.
266
267 ** Outputs: None
268
269 ** Return Codes:
270     ** None
271
272 ** Purpose: Save class data to a stream.
273
274
275
276
277 void
278 EDMResSubItemElement::saveOutputs(IN RWostream &strm)
279 {
280     // Save parent class data too
281     RWCollectable::saveOutputs(strm);
282
283     // Left as an exercise
284 }
285
286 /*****
287
288 ** Routine: restoreOutputs
289
290 ** Inputs:  RWFile f - file to read internal data from.
291
292 ** Outputs: None
293
294 ** Return Codes:
295     ** None
296
297 ** Purpose: Restores an instance of the Handle class by reading the
298             data from the passed in file.
299
300
301
302
303 void
304 EDMResSubItemElement::restoreOutputs(IN RWFile &f)
305 {
306     // Restore parent data too
307     RWCollectable::restoreOutputs(f);
308
309     // Left as an exercise
310 }
311
312 /*****

```

```

313
314 ** Routine: restoreOutputs
315
316 ** Inputs:  RWInputStream strm - stream to read internal data from.
317
318 ** Outputs: None
319
320 ** Return Codes:
321     ** None
322
323 ** Purpose: Restores an instance of the Handle class by reading the
324             data from the passed in stream.
325
326
327
328
329 void
330 EDMResSubItemElement::restoreOutputs(IN RWInputStream &strm)
331 {
332     // Restore parent data too
333     RWCollectable::restoreOutputs(strm);
334
335     // Left as an exercise
336 }
337
338 /*****
339
340 ** Routine: binaryStoreSize
341
342 ** Inputs:  None
343
344 ** Outputs: None
345
346 ** Return Codes:
347     ** Rhspace count - file size of class written to disk in
348                     bytes
349
350 ** Purpose: Returns the size of class if it were stored on disk.
351
352
353
354
355 Rhspace
356 EDMResSubItemElement::binaryStoreSize() const
357 {
358     Rhspace count = RWCollectable::binaryStoreSize() + 1;
359
360     // Left as an exercise
361     return count;
362 }
363
364 /*****
365
366 ** Routine: getSubItemElementID
367
368 ** Inputs:  None
369
370 ** Outputs: None
371
372 ** Return Codes:
373     ** None
374
375
376
377

```

```

373      The submit element ID
374      **
375      ** Purpose: Returns the submit element ID.
376      **           Will be 0 if not initialized fully.
377      **
378      .....
379
380      */
381
382      int
383      EDRSubItemElement::getSubItemElementID()
384      {
385          return subItemElementID;
386      }
387
388      .....
389
390      ** Routine: setSubItemElementID
391      **
392      ** Inputs:  int se_id - the ID for the submit element.
393      **           being inserted in the list.
394      **
395      ** Outputs: None
396      **
397      ** Return Codes:
398      **           None
399      **
400      ** Purpose: Sets the submit element ID.
401      **
402      .....
403
404      void
405      EDRSubItemElement::setSubItemElementID(int se_id)
406      {
407          subItemElementID = se_id;
408      }
409
410      .....
411
412      ** Routine: getWorkItem
413      **
414      ** Inputs:  int maxsize - max size to copy into buffer
415      **
416      ** Outputs: char *workItem - place to put workItem
417      **
418      ** Return Codes:
419      **           None
420      **
421      ** Purpose: Returns the work item name in the specified buffer.
422      **
423      .....
424
425      */
426
427      void
428      EDRSubItemElement::getWorkItem(char *workItem, int maxsize)
429      {
430          if (workItem == NULL && se_work_item != NULL)
431              strcpy(workItem, se_work_item, maxsize - 1);
432      }
433
434      .....
435
436      #ifdef _LIBS_restoreEDWRESsubmitElement.cc 7
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493

```

```

432      }
433      }
434      else if (workItem != NULL)
435      {
436          workItem[0] = 0;
437      }
438      .....
439
440      .....
441
442      ** Routine: setWorkItem
443      **
444      ** Inputs:  char *workItem - the workItem to copy
445      **
446      ** Outputs: None
447      **
448      ** Return Codes:
449      **           None
450      **
451      ** Purpose: Sets the work item name using the specified buffer.
452      **
453      .....
454
455      */
456
457      void
458      EDRSubItemElement::setWorkItem(char *workItem)
459      {
460          if (workItem != NULL)
461              if (se_work_item != NULL)
462                  free(se_work_item);
463          se_work_item = strdup(workItem);
464      }
465
466      .....
467
468      ** Routine: getTemplate
469      **
470      ** Inputs:  int maxsize - max size to copy into buffer
471      **
472      ** Outputs: char *template_name - place to put template name
473      **
474      ** Return Codes:
475      **           None
476      **
477      ** Purpose: Returns the template name in the specified buffer.
478      **
479      .....
480
481      .....
482
483      */
484
485      void
486      EDRSubItemElement::getTemplate(char *template_name, int maxsize)
487      {
488          if (template_name != NULL && se_template != NULL)
489              strcpy(template_name, se_template, maxsize - 1);
490          else if (template_name != NULL)
491              template_name[maxsize - 1] = 0;
492      }
493
494      .....
495
496      #ifdef _LIBS_restoreEDWRESsubmitElement.cc 8
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000

```

```

494 2      template_name[0] = 0;
495 1    }
496    }
497    /*****
498    **
499    ** Routine: settemplate
500    ** Inputs:  char *template_name - the template to copy
501    ** Outputs: None
502    ** Return Codes:
503    **      None
504    ** Purpose: Sets the template name using the specified buffer.
505    *****/
506    settemplate(char *template_name)
507    {
508      if (template_name == NULL)
509      {
510        if (se_template != NULL)
511          free(se_template);
512        se_template = strdup(template_name);
513      }
514    }
515    /*****
516    **
517    ** Routine: isalternatetemplate
518    ** Inputs:  None
519    ** Outputs: None
520    ** Return Codes:
521    **      Boolean value of se_is_template_alternate element
522    ** Purpose: Returns the TRUE if alternate template used.
523    *****/
524    isalternatetemplate()
525    {
526      return se_is_template_alternate;
527    }
528    /*****
529    **
530    ** Routine: SetAlternateTemplate
531    ** Inputs:  boolean_t - set it to TRUE or FALSE
532    ** Outputs: None
533    *****/
534    SetAlternateTemplate(boolean_t alternate)
535    {
536      se_is_template_alternate = alternate;
537    }
538    /*****
539    **
540    ** Routine: SetAlternateTemplate
541    ** Inputs:  char *template_name - the template to copy
542    ** Outputs: None
543    ** Return Codes:
544    **      None
545    ** Purpose: Sets the template name using the specified buffer.
546    *****/
547    settemplate(char *template_name)
548    {
549      if (template_name == NULL)
550      {
551        if (se_template != NULL)
552          free(se_template);
553        se_template = strdup(template_name);
554      }
555    }
556    /*****
557    **
558    ** Routine: isalternatetemplate
559    ** Inputs:  None
560    ** Outputs: None
561    ** Return Codes:
562    **      Boolean value of se_is_template_alternate element
563    ** Purpose: Returns the TRUE if alternate template used.
564    *****/
565    isalternatetemplate()
566    {
567      return se_is_template_alternate;
568    }
569    /*****
570    **
571    ** Routine: SetAlternateTemplate
572    ** Inputs:  boolean_t - set it to TRUE or FALSE
573    ** Outputs: None
574    *****/
575    SetAlternateTemplate(boolean_t alternate)
576    {
577      se_is_template_alternate = alternate;
578    }
579    /*****
580    **
581    ** Routine: SetAlternateTemplate
582    ** Inputs:  char *template_name - the template to copy
583    ** Outputs: None
584    ** Return Codes:
585    **      None
586    ** Purpose: Sets the template name using the specified buffer.
587    *****/
588    settemplate(char *template_name)
589    {
590      if (template_name == NULL)
591      {
592        if (se_template != NULL)
593          free(se_template);
594        se_template = strdup(template_name);
595      }
596    }
597    /*****
598    **
599    ** Routine: isalternatetemplate
600    ** Inputs:  None
601    ** Outputs: None
602    ** Return Codes:
603    **      Boolean value of se_is_template_alternate element
604    ** Purpose: Returns the TRUE if alternate template used.
605    *****/
606    isalternatetemplate()
607    {
608      return se_is_template_alternate;
609    }
610    /*****
611    **
612    ** Routine: SetAlternateTemplate
613    ** Inputs:  boolean_t - set it to TRUE or FALSE
614    ** Outputs: None
615    *****/
616    SetAlternateTemplate(boolean_t alternate)
617    {
618      se_is_template_alternate = alternate;
619    }
620    /*****
621    **
622    ** Routine: SetAlternateTemplate
623    ** Inputs:  char *template_name - the template to copy
624    ** Outputs: None
625    ** Return Codes:
626    **      None
627    ** Purpose: Sets the template name using the specified buffer.
628    *****/
629    settemplate(char *template_name)
630    {
631      if (template_name == NULL)
632      {
633        if (se_template != NULL)
634          free(se_template);
635        se_template = strdup(template_name);
636      }
637    }
638    /*****
639    **
640    ** Routine: isalternatetemplate
641    ** Inputs:  None
642    ** Outputs: None
643    ** Return Codes:
644    **      Boolean value of se_is_template_alternate element
645    ** Purpose: Returns the TRUE if alternate template used.
646    *****/
647    isalternatetemplate()
648    {
649      return se_is_template_alternate;
650    }
651    /*****
652    **
653    ** Routine: SetAlternateTemplate
654    ** Inputs:  boolean_t - set it to TRUE or FALSE
655    ** Outputs: None
656    *****/
657    SetAlternateTemplate(boolean_t alternate)
658    {
659      se_is_template_alternate = alternate;
660    }
661    /*****
662    **
663    ** Routine: SetAlternateTemplate
664    ** Inputs:  char *template_name - the template to copy
665    ** Outputs: None
666    ** Return Codes:
667    **      None
668    ** Purpose: Sets the template name using the specified buffer.
669    *****/
670    settemplate(char *template_name)
671    {
672      if (template_name == NULL)
673      {
674        if (se_template != NULL)
675          free(se_template);
676        se_template = strdup(template_name);
677      }
678    }
679    /*****
680    **
681    ** Routine: isalternatetemplate
682    ** Inputs:  None
683    ** Outputs: None
684    ** Return Codes:
685    **      Boolean value of se_is_template_alternate element
686    ** Purpose: Returns the TRUE if alternate template used.
687    *****/
688    isalternatetemplate()
689    {
690      return se_is_template_alternate;
691    }
692    /*****
693    **
694    ** Routine: SetAlternateTemplate
695    ** Inputs:  boolean_t - set it to TRUE or FALSE
696    ** Outputs: None
697    *****/
698    SetAlternateTemplate(boolean_t alternate)
699    {
700      se_is_template_alternate = alternate;
701    }
702    /*****
703    **
704    ** Routine: SetAlternateTemplate
705    ** Inputs:  char *template_name - the template to copy
706    ** Outputs: None
707    ** Return Codes:
708    **      None
709    ** Purpose: Sets the template name using the specified buffer.
710    *****/
711    settemplate(char *template_name)
712    {
713      if (template_name == NULL)
714      {
715        if (se_template != NULL)
716          free(se_template);
717        se_template = strdup(template_name);
718      }
719    }
720    /*****
721    **
722    ** Routine: isalternatetemplate
723    ** Inputs:  None
724    ** Outputs: None
725    ** Return Codes:
726    **      Boolean value of se_is_template_alternate element
727    ** Purpose: Returns the TRUE if alternate template used.
728    *****/
729    isalternatetemplate()
730    {
731      return se_is_template_alternate;
732    }
733    /*****
734    **
735    ** Routine: SetAlternateTemplate
736    ** Inputs:  boolean_t - set it to TRUE or FALSE
737    ** Outputs: None
738    *****/
739    SetAlternateTemplate(boolean_t alternate)
740    {
741      se_is_template_alternate = alternate;
742    }
743    /*****
744    **
745    ** Routine: SetAlternateTemplate
746    ** Inputs:  char *template_name - the template to copy
747    ** Outputs: None
748    ** Return Codes:
749    **      None
750    ** Purpose: Sets the template name using the specified buffer.
751    *****/
752    settemplate(char *template_name)
753    {
754      if (template_name == NULL)
755      {
756        if (se_template != NULL)
757          free(se_template);
758        se_template = strdup(template_name);
759      }
760    }
761    /*****
762    **
763    ** Routine: isalternatetemplate
764    ** Inputs:  None
765    ** Outputs: None
766    ** Return Codes:
767    **      Boolean value of se_is_template_alternate element
768    ** Purpose: Returns the TRUE if alternate template used.
769    *****/
770    isalternatetemplate()
771    {
772      return se_is_template_alternate;
773    }
774    /*****
775    **
776    ** Routine: SetAlternateTemplate
777    ** Inputs:  boolean_t - set it to TRUE or FALSE
778    ** Outputs: None
779    *****/
780    SetAlternateTemplate(boolean_t alternate)
781    {
782      se_is_template_alternate = alternate;
783    }
784    /*****
785    **
786    ** Routine: SetAlternateTemplate
787    ** Inputs:  char *template_name - the template to copy
788    ** Outputs: None
789    ** Return Codes:
790    **      None
791    ** Purpose: Sets the template name using the specified buffer.
792    *****/
793    settemplate(char *template_name)
794    {
795      if (template_name == NULL)
796      {
797        if (se_template != NULL)
798          free(se_template);
799        se_template = strdup(template_name);
800      }
801    }
802    /*****
803    **
804    ** Routine: isalternatetemplate
805    ** Inputs:  None
806    ** Outputs: None
807    ** Return Codes:
808    **      Boolean value of se_is_template_alternate element
809    ** Purpose: Returns the TRUE if alternate template used.
810    *****/
811    isalternatetemplate()
812    {
813      return se_is_template_alternate;
814    }
815    /*****
816    **
817    ** Routine: SetAlternateTemplate
818    ** Inputs:  boolean_t - set it to TRUE or FALSE
819    ** Outputs: None
820    *****/
821    SetAlternateTemplate(boolean_t alternate)
822    {
823      se_is_template_alternate = alternate;
824    }
825    /*****
826    **
827    ** Routine: SetAlternateTemplate
828    ** Inputs:  char *template_name - the template to copy
829    ** Outputs: None
830    ** Return Codes:
831    **      None
832    ** Purpose: Sets the template name using the specified buffer.
833    *****/
834    settemplate(char *template_name)
835    {
836      if (template_name == NULL)
837      {
838        if (se_template != NULL)
839          free(se_template);
840        se_template = strdup(template_name);
841      }
842    }
843    /*****
844    **
845    ** Routine: isalternatetemplate
846    ** Inputs:  None
847    ** Outputs: None
848    ** Return Codes:
849    **      Boolean value of se_is_template_alternate element
850    ** Purpose: Returns the TRUE if alternate template used.
851    *****/
852    isalternatetemplate()
853    {
854      return se_is_template_alternate;
855    }
856    /*****
857    **
858    ** Routine: SetAlternateTemplate
859    ** Inputs:  boolean_t - set it to TRUE or FALSE
860    ** Outputs: None
861    *****/
862    SetAlternateTemplate(boolean_t alternate)
863    {
864      se_is_template_alternate = alternate;
865    }
866    /*****
867    **
868    ** Routine: SetAlternateTemplate
869    ** Inputs:  char *template_name - the template to copy
870    ** Outputs: None
871    ** Return Codes:
872    **      None
873    ** Purpose: Sets the template name using the specified buffer.
874    *****/
875    settemplate(char *template_name)
876    {
877      if (template_name == NULL)
878      {
879        if (se_template != NULL)
880          free(se_template);
881        se_template = strdup(template_name);
882      }
883    }
884    /*****
885    **
886    ** Routine: isalternatetemplate
887    ** Inputs:  None
888    ** Outputs: None
889    ** Return Codes:
890    **      Boolean value of se_is_template_alternate element
891    ** Purpose: Returns the TRUE if alternate template used.
892    *****/
893    isalternatetemplate()
894    {
895      return se_is_template_alternate;
896    }
897    /*****
898    **
899    ** Routine: SetAlternateTemplate
900    ** Inputs:  boolean_t - set it to TRUE or FALSE
901    ** Outputs: None
902    *****/
903    SetAlternateTemplate(boolean_t alternate)
904    {
905      se_is_template_alternate = alternate;
906    }
907    /*****
908    **
909    ** Routine: SetAlternateTemplate
910    ** Inputs:  char *template_name - the template to copy
911    ** Outputs: None
912    ** Return Codes:
913    **      None
914    ** Purpose: Sets the template name using the specified buffer.
915    *****/
916    settemplate(char *template_name)
917    {
918      if (template_name == NULL)
919      {
920        if (se_template != NULL)
921         
```

535	**	Return Codes:	
536	**	None	
537	**		
538	**		
539	**	Purpose: Sets the se_is_trailer_alternate variable to the given parameter	
540	**		
541	**		
542	**		
543	**		
544	**		
545	**		
546	**		
547	**		
548	**		
549	**		
550	**		
551	**		
552	**		
553	**		
554	**		
555	**		
556	**		
557	**		
558	**		
559	**		
560	**		
561	**		
562	**		
563	**		
564	**		
565	**		
566	**		
567	**		
568	**		
569	**		
570	**		
571	**		
572	**		
573	**		
574	**		
575	**		
576	**		
577	**		
578	**		
579	**		
580	**		
581	**		
582	**		
583	**		
584	**		
585	**		
586	**		
587	**		
588	**		
589	**		
590	**		
591	**		
592	**		
593	**		
594	**		
595	**		
596	**		
597	**		
598	**		
599	**		
600	**		
601	**		
602	**		
603	**		
604	**		
605	**		
606	**		
607	**		
608	**		
609	**		
610	**		
611	**		
612	**		
613	**		

614	*/		
616	void		
617	EDMRSubItemElement::setClientSourceName(char *clientname)		
618	{		
619	if (clientname != NULL)		
620	{		
621	if (se_client_rbufname != NULL)		
622	free(se_client_rbufname);		
623	se_client_rbufname = strdup(clientname);		
624	}		
625	}		
626			
628		
629	*/		
630	ROUTINE: getClientSource		
631	ROUTINE: int maxsize - max size to copy into buffer		
632	INPUTS: int maxsize - max size to copy into buffer		
633	OUTPUTS: char *clientname - place to put client that was backed up		
634	Return Codes:		
635	None		
636	Purpose: Returns the client name in the specified buffer.		
637		
638	*/		
639	void		
640	EDMRSubItemElement::getClientSource(char *clientname, int maxsize)		
641	{		
642	if (clientname != NULL && se_source_client_hostname != NULL)		
643	{		
644	strcpy(clientname, se_source_client_hostname, maxsize - 1);		
645	clientname[maxsize - 1] = 0;		
646	} else if (clientname != NULL)		
647	{		
648	clientname[0] = 0;		
649	}		
650	}		
651		
652	ROUTINE: sectionSource		
653	ROUTINE: char *clientname - the client that was backed up		
654	INPUTS: char *clientname - the client that was backed up		
655	OUTPUTS: None		
656	Return Codes:		
657	None		
658	Purpose: Sets the client name that was backed up using the		
659	specified buffer.		
660		
661	*/		
662	void		
663	EDMRSubItemElement::setClientSourceName(char *clientname)		
664	{		
665	if (clientname != NULL)		
666	{		
667	if (se_client_rbufname != NULL)		
668	free(se_client_rbufname);		
669	se_client_rbufname = strdup(clientname);		
670	}		
671	}		
672			
673		
674	*/		
675	void		
676	EDMRSubItemElement::setClientSourceName(char *clientname)		
677	{		
678	if (clientname != NULL)		
679	{		
680	if (se_client_rbufname != NULL)		
681	free(se_client_rbufname);		
682	se_client_rbufname = strdup(clientname);		
683	}		
684	}		
685			
686		
687	ROUTINE: getClientHostname		
688	ROUTINE: int maxsize - max size to copy into buffer		
689	INPUTS: int maxsize - max size to copy into buffer		
690	OUTPUTS: char *clienthost - place to put the client hostname		
691	Return Codes:		
692	None		
693	Purpose: Returns the client host name in the specified buffer.		
694		
695	*/		
696	void		
697	EDMRSubItemElement::getClientHost(char *clienthost, int maxsize)		
698	{		
699	if (clienthost != NULL && se_client_hostname != NULL)		
700	{		
701	strcpy(clienthost, se_client_hostname, maxsize - 1);		
702	clienthost[maxsize - 1] = 0;		
703	} else if (clienthost != NULL)		
704	{		
705	clienthost[0] = 0;		
706	}		
707	}		
708			
709		
710	ROUTINE: sectionHostname		
711	ROUTINE: char *clienthost - the client hostname		
712	INPUTS: char *clienthost - the client hostname		
713	OUTPUTS: None		
714	Return Codes:		
715	None		
716	Purpose: Sets the client host name using the specified buffer.		
717		
718	*/		
719	void		
720	EDMRSubItemElement::setClientHost(char *clienthost)		
721	{		
722	if (clienthost != NULL)		
723	{		
724	if (se_client_rbufname != NULL)		
725	free(se_client_rbufname);		
726	se_client_rbufname = strdup(clienthost);		
727	}		
728	}		
729			
730		
731	*/		
732	void		
733	EDMRSubItemElement::setClientHost(char *clienthost)		
734	{		
735	if (clienthost != NULL)		
736	{		
737	if (se_client_rbufname != NULL)		
738	free(se_client_rbufname);		
739	se_client_rbufname = strdup(clienthost);		
740	}		
741	}		
742			
743		
744	*/		

616	EDMRSubItemElement::setClientSource(char *clientname)		
617	{		
618	if (clientname != NULL)		
619	{		
620	if (se_source_client_hostname != NULL)		
621	free(se_source_client_hostname);		
622	se_source_client_hostname = strdup(clientname);		
623	}		
624	}		
625			
626		
627	ROUTINE: getClientHostname		
628	ROUTINE: int maxsize - max size to copy into buffer		
629	INPUTS: int maxsize - max size to copy into buffer		
630	OUTPUTS: char *clienthost - place to put the client hostname		
631	Return Codes:		
632	None		
633	Purpose: Returns the client host name in the specified buffer.		
634		
635	*/		
636	void		
637	EDMRSubItemElement::getClientHost(char *clienthost, int maxsize)		
638	{		
639	if (clienthost != NULL && se_client_hostname != NULL)		
640	{		
641	strcpy(clienthost, se_client_hostname, maxsize - 1);		
642	clienthost[maxsize - 1] = 0;		
643	} else if (clienthost != NULL)		
644	{		
645	clienthost[0] = 0;		
646	}		
647	}		
648			
649		
650	ROUTINE: sectionHostname		
651	ROUTINE: char *clienthost - the client hostname		
652	INPUTS: char *clienthost - the client hostname		
653	OUTPUTS: None		
654	Return Codes:		
655	None		
656	Purpose: Sets the client host name using the specified buffer.		
657		
658	*/		
659	void		
660	EDMRSubItemElement::setClientHost(char *clienthost)		
661	{		
662	if (clienthost != NULL)		
663	{		
664	if (se_client_rbufname != NULL)		
665	free(se_client_rbufname);		
666	se_client_rbufname = strdup(clienthost);		
667	}		
668	}		
669			
670		
671	*/		

```

738 2         if (se_client_hostname != NULL)
739 2             free(se_client_hostname);
740 2         se_client_hostname = strdup(c1len*host);
741 2     }
742 1 }
743 1 }
744 1 }
745 1 }
746 1 }
747 1 }
748 1 }
749 1 }
750 1 }
751 1 }
752 1 }
753 1 }
754 1 }
755 1 }
756 1 }
757 1 }
758 1 }
759 1 }
760 1 }
761 1 }
762 1 }
763 1 }
764 1 }
765 1 }
766 1 }
767 1 }
768 1 }
769 1 }
770 1 }
771 1 }
772 1 }
773 1 }
774 1 }
775 1 }
776 1 }
777 1 }
778 1 }
779 1 }
780 1 }
781 1 }
782 1 }
783 1 }
784 1 }
785 1 }
786 1 }
787 1 }
788 1 }
789 1 }
790 1 }
791 1 }
792 1 }
793 1 }
794 1 }
795 1 }
796 1 }
797 1 }
798 1 }
799 1 }
800 1 }
801 1 }
802 1 }
803 1 }
804 1 }
805 1 }
806 1 }
807 1 }
808 1 }
809 1 }
810 1 }
811 1 }
812 1 }
813 1 }
814 1 }
815 1 }
816 1 }
817 1 }
818 1 }
819 1 }
820 1 }
821 1 }
822 1 }
823 1 }
824 1 }
825 1 }
826 1 }
827 1 }
828 1 }
829 1 }
830 1 }
831 1 }
832 1 }
833 1 }
834 1 }
835 1 }
836 1 }
837 1 }
838 1 }
839 1 }
840 1 }
841 1 }
842 1 }
843 1 }
844 1 }
845 1 }
846 1 }
847 1 }
848 1 }
849 1 }
850 1 }
851 1 }
852 1 }
853 1 }
854 1 }
855 1 }
856 1 }
857 1 }
858 1 }
859 1 }
860 1 }
861 1 }
862 1 }
863 1 }
864 1 }
865 1 }
866 1 }
867 1 }
868 1 }
869 1 }
870 1 }
871 1 }
872 1 }
873 1 }
874 1 }
875 1 }
876 1 }
877 1 }
878 1 }
879 1 }
880 1 }
881 1 }
882 1 }
883 1 }
884 1 }
885 1 }
886 1 }
887 1 }
888 1 }
889 1 }
890 1 }
891 1 }
892 1 }
893 1 }
894 1 }
895 1 }
896 1 }
897 1 }
898 1 }
899 1 }
900 1 }
901 1 }
902 1 }
903 1 }
904 1 }
905 1 }
906 1 }
907 1 }
908 1 }
909 1 }
910 1 }
911 1 }
912 1 }
913 1 }
914 1 }
915 1 }
916 1 }
917 1 }
918 1 }
919 1 }
920 1 }
921 1 }
922 1 }
923 1 }
924 1 }
925 1 }
926 1 }
927 1 }
928 1 }
929 1 }
930 1 }
931 1 }
932 1 }
933 1 }
934 1 }
935 1 }
936 1 }
937 1 }
938 1 }
939 1 }
940 1 }
941 1 }
942 1 }
943 1 }
944 1 }
945 1 }
946 1 }
947 1 }
948 1 }
949 1 }
950 1 }
951 1 }
952 1 }
953 1 }
954 1 }
955 1 }
956 1 }
957 1 }
958 1 }
959 1 }
960 1 }
961 1 }
962 1 }
963 1 }
964 1 }
965 1 }
966 1 }
967 1 }
968 1 }
969 1 }
970 1 }
971 1 }
972 1 }
973 1 }
974 1 }
975 1 }
976 1 }
977 1 }
978 1 }
979 1 }
980 1 }
981 1 }
982 1 }
983 1 }
984 1 }
985 1 }
986 1 }
987 1 }
988 1 }
989 1 }
990 1 }
991 1 }
992 1 }
993 1 }
994 1 }
995 1 }
996 1 }
997 1 }
998 1 }
999 1 }
1000 1 }

```

```

799 1 }
800 1 }
801 1 }
802 1 }
803 1 }
804 1 }
805 1 }
806 1 }
807 1 }
808 1 }
809 1 }
810 1 }
811 1 }
812 1 }
813 1 }
814 1 }
815 1 }
816 1 }
817 1 }
818 1 }
819 1 }
820 1 }
821 1 }
822 1 }
823 1 }
824 1 }
825 1 }
826 1 }
827 1 }
828 1 }
829 1 }
830 1 }
831 1 }
832 1 }
833 1 }
834 1 }
835 1 }
836 1 }
837 1 }
838 1 }
839 1 }
840 1 }
841 1 }
842 1 }
843 1 }
844 1 }
845 1 }
846 1 }
847 1 }
848 1 }
849 1 }
850 1 }
851 1 }
852 1 }
853 1 }
854 1 }
855 1 }
856 1 }
857 1 }
858 1 }
859 1 }
860 1 }
861 1 }
862 1 }
863 1 }
864 1 }
865 1 }
866 1 }
867 1 }
868 1 }
869 1 }
870 1 }
871 1 }
872 1 }
873 1 }
874 1 }
875 1 }
876 1 }
877 1 }
878 1 }
879 1 }
880 1 }
881 1 }
882 1 }
883 1 }
884 1 }
885 1 }
886 1 }
887 1 }
888 1 }
889 1 }
890 1 }
891 1 }
892 1 }
893 1 }
894 1 }
895 1 }
896 1 }
897 1 }
898 1 }
899 1 }
900 1 }
901 1 }
902 1 }
903 1 }
904 1 }
905 1 }
906 1 }
907 1 }
908 1 }
909 1 }
910 1 }
911 1 }
912 1 }
913 1 }
914 1 }
915 1 }
916 1 }
917 1 }
918 1 }
919 1 }
920 1 }
921 1 }
922 1 }
923 1 }
924 1 }
925 1 }
926 1 }
927 1 }
928 1 }
929 1 }
930 1 }
931 1 }
932 1 }
933 1 }
934 1 }
935 1 }
936 1 }
937 1 }
938 1 }
939 1 }
940 1 }
941 1 }
942 1 }
943 1 }
944 1 }
945 1 }
946 1 }
947 1 }
948 1 }
949 1 }
950 1 }
951 1 }
952 1 }
953 1 }
954 1 }
955 1 }
956 1 }
957 1 }
958 1 }
959 1 }
960 1 }
961 1 }
962 1 }
963 1 }
964 1 }
965 1 }
966 1 }
967 1 }
968 1 }
969 1 }
970 1 }
971 1 }
972 1 }
973 1 }
974 1 }
975 1 }
976 1 }
977 1 }
978 1 }
979 1 }
980 1 }
981 1 }
982 1 }
983 1 }
984 1 }
985 1 }
986 1 }
987 1 }
988 1 }
989 1 }
990 1 }
991 1 }
992 1 }
993 1 }
994 1 }
995 1 }
996 1 }
997 1 }
998 1 }
999 1 }
1000 1 }

```

```

859 ** Purpose: Returns the directory in the specified buffer.
860 **
861 *****
862 */
863 void
864 EKMSesSubItemElement::getDirectoryDestination(
865     char *directory, int maxsize)
866 {
867     if (directory != NULL && se_directory != NULL)
868     {
869         strncpy(directory, se_directory, maxsize - 1);
870         directory[maxsize - 1] = 0;
871     }
872     else if (directory != NULL)
873     {
874         directory[0] = 0;
875     }
876 }
877
878 /*****
879 **
880 ** Routine: seDirectoryDestination
881 **
882 ** Inputs: char *directory - place to put the restore
883 **          Outputs: None
884 **
885 ** Return Codes:
886 **      None
887 **
888 ** Purpose: Sets the destination for the restore using the specified
889 **          buffer.
890 **
891 *****/
892 */
893 void
894 EKMSesSubItemElement::setDirectoryDestination(char *directory)
895 {
896     if (directory != NULL)
897     {
898         if (se_directory != NULL)
899             free(se_directory);
900         se_directory = strdup(directory);
901     }
902 }
903
904 /*****
905 **
906 ** Routine: getRestoreTransport
907 **
908 ** Inputs: None
909 **
910 ** Outputs: None
911 **
912 ** Return Codes:
913 **      RestoreTransport - as enumerated in restore_api.h
914 **
915 ** Purpose: Returns the restore transport
916 **
917 *****/

```

```

919 *****
920 */
921 RestoreTransport
922 EKMSesSubItemElement::getRestoreTransport()
923 {
924     return se_transport;
925 }
926
927 /*****
928 **
929 ** Routine: setRestoreTransport
930 **
931 ** Inputs: RestoreTransport transport - use this to set the restore
932 **          transport
933 **
934 ** Outputs: None
935 **
936 ** Return Codes:
937 **      None
938 **
939 ** Purpose: Sets the restore transport for the restore using the
940 **          specified parameter.
941 **
942 *****/
943 */
944 void
945 EKMSesSubItemElement::setRestoreTransport(RestoreTransport transport)
946 {
947     se_transport = transport;
948 }
949
950 /*****
951 **
952 ** Routine: getOverwritePolicy
953 **
954 ** Inputs: None
955 **
956 ** Outputs: None
957 **
958 ** Return Codes:
959 **      OverwritePolicy - as enumerated in restore_api.h
960 **
961 ** Purpose: Returns the overwrite policy
962 **
963 *****/
964 */
965 OverwritePolicy
966 EKMSesSubItemElement::getOverwritePolicy()
967 {
968     return se_policy;
969 }
970
971 /*****
972 **
973 ** Routine: setOverwritePolicy
974 **
975 ** Inputs: OverwritePolicy policy - use this to set the overwrite
976 **
977 *****/

```

```

978      **
979      ** Outputs:  None
980      **
981      ** Return Codes:
982      **      None
983      **
984      ** Purpose: Sets the overwrite policy for the restore using the
985      **      specified parameter.
986      **
987      **
988      **
989      */
990      void
991      EDRessubmitElement::setOverwritePolicy(OverwritePolicy policy)
992      {
993          se_policy = policy;
994      }
995      /*****
996
997      ** Routine: getScriptName
998      **
999      ** Inputs:  int maxsize - max size to copy into buffer
1000      **
1001      ** Outputs: char *scriptname - place to put the script that will run
1002      **          on client
1003      **
1004      ** Return Codes:
1005      **      None
1006      **
1007      ** Purpose: Returns the script name in the specified buffer.
1008      **
1009      *****/
1010      void
1011      EDRessubmitElement::getScriptName(char *scriptname, int maxlen)
1012      {
1013          if (scriptname == NULL && se_client_scriptname != NULL)
1014          {
1015              strncpy(scriptname, se_client_scriptname, maxlen - 1);
1016              scriptname[maxlen - 1] = 0;
1017          }
1018          else if (scriptname != NULL)
1019          {
1020              scriptname[0] = 0;
1021          }
1022      }
1023      /*****
1024
1025      ** Routine: setScriptName
1026      **
1027      ** Inputs:  char *scriptname - script to run on the client
1028      **
1029      ** Outputs: None
1030      **
1031      ** Return Codes:
1032      **      None
1033      **
1034      ** Purpose: Sets the scriptname to be used on the client using the
1035      **      specified buffer.
1036      *****/
1037      void
1038      EDRessubmitElement::setScriptName(char *scriptname)
1039      {
1040          if (scriptname != NULL)
1041          {
1042              if (se_client_scriptname != NULL)
1043              {
1044                  free(se_client_scriptname);
1045              }
1046              se_client_scriptname = strdup(scriptname);
1047          }
1048      }
1049      /*****
1050
1051      ** Routine: getSocketHost
1052      **
1053      ** Inputs:  int maxsize - max size to copy into buffer
1054      **
1055      ** Outputs: char *sockethost - place to put hostname to connect to
1056      **
1057      ** Return Codes:
1058      **      None
1059      **
1060      ** Purpose: Returns the hostname to use in the specified buffer.
1061      **
1062      *****/
1063      void
1064      EDRessubmitElement::getSocketHost(char *sockethost, int maxsize)
1065      {
1066          if (sockethost != NULL && se_socket_host != NULL)
1067          {
1068              strncpy(sockethost, se_socket_host, maxsize - 1);
1069              sockethost[maxsize - 1] = 0;
1070          }
1071          else if (sockethost != NULL)
1072          {
1073              sockethost[0] = 0;
1074          }
1075      }
1076      /*****
1077
1078      ** Routine: setSocketHost
1079      **
1080      ** Inputs:  char *sockethost - hostname to connect to
1081      **
1082      ** Outputs: None
1083      **
1084      ** Return Codes:
1085      **      None
1086      **
1087      ** Purpose: Sets the hostname to connect to using the specified
1088      **      buffer.
1089      *****/
1090      void
1091      EDRessubmitElement::setSocketHost(char *sockethost)
1092      {
1093          if (sockethost != NULL)
1094          {
1095              if (se_socket_host != NULL)
1096              {
1097                  free(se_socket_host);
1098              }
1099              se_socket_host = strdup(sockethost);
1100          }
1101      }
1102      /*****
1103
1104      ** Routine: setScriptName
1105      **
1106      ** Inputs:  char *scriptname - script to run on the client
1107      **
1108      ** Outputs: None
1109      **
1110      ** Return Codes:
1111      **      None
1112      **
1113      ** Purpose: Sets the scriptname to be used on the client using the
1114      **      specified buffer.
1115      *****/
1116      void
1117      EDRessubmitElement::setScriptName(char *scriptname)
1118      {
1119          if (scriptname != NULL)
1120          {
1121              if (se_client_scriptname != NULL)
1122              {
1123                  free(se_client_scriptname);
1124              }
1125              se_client_scriptname = strdup(scriptname);
1126          }
1127      }
1128      /*****
1129
1130      ** Routine: setSocketHost
1131      **
1132      ** Inputs:  char *sockethost - hostname to connect to
1133      **
1134      ** Outputs: None
1135      **
1136      ** Return Codes:
1137      **      None
1138      **
1139      ** Purpose: Sets the hostname to connect to using the specified
1140      **      buffer.
1141      *****/
1142      void
1143      EDRessubmitElement::setSocketHost(char *sockethost)
1144      {
1145          if (sockethost != NULL)
1146          {
1147              if (se_socket_host != NULL)
1148              {
1149                  free(se_socket_host);
1150              }
1151              se_socket_host = strdup(sockethost);
1152          }
1153      }
1154      /*****
1155
1156      ** Routine: setScriptName
1157      **
1158      ** Inputs:  char *scriptname - script to run on the client
1159      **
1160      ** Outputs: None
1161      **
1162      ** Return Codes:
1163      **      None
1164      **
1165      ** Purpose: Sets the scriptname to be used on the client using the
1166      **      specified buffer.
1167      *****/
1168      void
1169      EDRessubmitElement::setScriptName(char *scriptname)
1170      {
1171          if (scriptname != NULL)
1172          {
1173              if (se_client_scriptname != NULL)
1174              {
1175                  free(se_client_scriptname);
1176              }
1177              se_client_scriptname = strdup(scriptname);
1178          }
1179      }
1180      /*****
1181
1182      ** Routine: setSocketHost
1183      **
1184      ** Inputs:  char *sockethost - hostname to connect to
1185      **
1186      ** Outputs: None
1187      **
1188      ** Return Codes:
1189      **      None
1190      **
1191      ** Purpose: Sets the hostname to connect to using the specified
1192      **      buffer.
1193      *****/
1194      void
1195      EDRessubmitElement::setSocketHost(char *sockethost)
1196      {
1197          if (sockethost != NULL)
1198          {
1199              if (se_socket_host != NULL)
1200              {
1201                  free(se_socket_host);
1202              }
1203              se_socket_host = strdup(sockethost);
1204          }
1205      }
1206      /*****
1207
1208      ** Routine: setScriptName
1209      **
1210      ** Inputs:  char *scriptname - script to run on the client
1211      **
1212      ** Outputs: None
1213      **
1214      ** Return Codes:
1215      **      None
1216      **
1217      ** Purpose: Sets the scriptname to be used on the client using the
1218      **      specified buffer.
1219      *****/
1220      void
1221      EDRessubmitElement::setScriptName(char *scriptname)
1222      {
1223          if (scriptname != NULL)
1224          {
1225              if (se_client_scriptname != NULL)
1226              {
1227                  free(se_client_scriptname);
1228              }
1229              se_client_scriptname = strdup(scriptname);
1230          }
1231      }
1232      /*****
1233
1234      ** Routine: setSocketHost
1235      **
1236      ** Inputs:  char *sockethost - hostname to connect to
1237      **
1238      ** Outputs: None
1239      **
1240      ** Return Codes:
1241      **      None
1242      **
1243      ** Purpose: Sets the hostname to connect to using the specified
1244      **      buffer.
1245      *****/
1246      void
1247      EDRessubmitElement::setSocketHost(char *sockethost)
1248      {
1249          if (sockethost != NULL)
1250          {
1251              if (se_socket_host != NULL)
1252              {
1253                  free(se_socket_host);
1254              }
1255              se_socket_host = strdup(sockethost);
1256          }
1257      }
1258      /*****
1259
1260      ** Routine: setScriptName
1261      **
1262      ** Inputs:  char *scriptname - script to run on the client
1263      **
1264      ** Outputs: None
1265      **
1266      ** Return Codes:
1267      **      None
1268      **
1269      ** Purpose: Sets the scriptname to be used on the client using the
1270      **      specified buffer.
1271      *****/
1272      void
1273      EDRessubmitElement::setScriptName(char *scriptname)
1274      {
1275          if (scriptname != NULL)
1276          {
1277              if (se_client_scriptname != NULL)
1278              {
1279                  free(se_client_scriptname);
1280              }
1281              se_client_scriptname = strdup(scriptname);
1282          }
1283      }
1284      /*****
1285
1286      ** Routine: setSocketHost
1287      **
1288      ** Inputs:  char *sockethost - hostname to connect to
1289      **
1290      ** Outputs: None
1291      **
1292      ** Return Codes:
1293      **      None
1294      **
1295      ** Purpose: Sets the hostname to connect to using the specified
1296      **      buffer.
1297      *****/
1298      void
1299      EDRessubmitElement::setSocketHost(char *sockethost)
1300      {
1301          if (sockethost != NULL)
1302          {
1303              if (se_socket_host != NULL)
1304              {
1305                  free(se_socket_host);
1306              }
1307              se_socket_host = strdup(sockethost);
1308          }
1309      }
1310      /*****
1311
1312      ** Routine: setScriptName
1313      **
1314      ** Inputs:  char *scriptname - script to run on the client
1315      **
1316      ** Outputs: None
1317      **
1318      ** Return Codes:
1319      **      None
1320      **
1321      ** Purpose: Sets the scriptname to be used on the client using the
1322      **      specified buffer.
1323      *****/
1324      void
1325      EDRessubmitElement::setScriptName(char *scriptname)
1326      {
1327          if (scriptname != NULL)
1328          {
1329              if (se_client_scriptname != NULL)
1330              {
1331                  free(se_client_scriptname);
1332              }
1333              se_client_scriptname = strdup(scriptname);
1334          }
1335      }
1336      /*****
1337
1338      ** Routine: setSocketHost
1339      **
1340      ** Inputs:  char *sockethost - hostname to connect to
1341      **
1342      ** Outputs: None
1343      **
1344      ** Return Codes:
1345      **      None
1346      **
1347      ** Purpose: Sets the hostname to connect to using the specified
1348      **      buffer.
1349      *****/
1350      void
1351      EDRessubmitElement::setSocketHost(char *sockethost)
1352      {
1353          if (sockethost != NULL)
1354          {
1355              if (se_socket_host != NULL)
1356              {
1357                  free(se_socket_host);
1358              }
1359              se_socket_host = strdup(sockethost);
1360          }
1361      }
1362      /*****
1363
1364      ** Routine: setScriptName
1365      **
1366      ** Inputs:  char *scriptname - script to run on the client
1367      **
1368      ** Outputs: None
1369      **
1370      ** Return Codes:
1371      **      None
1372      **
1373      ** Purpose: Sets the scriptname to be used on the client using the
1374      **      specified buffer.
1375      *****/
1376      void
1377      EDRessubmitElement::setScriptName(char *scriptname)
1378      {
1379          if (scriptname != NULL)
1380          {
1381              if (se_client_scriptname != NULL)
1382              {
1383                  free(se_client_scriptname);
1384              }
1385              se_client_scriptname = strdup(scriptname);
1386          }
1387      }
1388      /*****
1389
1390      ** Routine: setSocketHost
1391      **
1392      ** Inputs:  char *sockethost - hostname to connect to
1393      **
1394      ** Outputs: None
1395      **
1396      ** Return Codes:
1397      **      None
1398      **
1399      ** Purpose: Sets the hostname to connect to using the specified
1400      **      buffer.
1401      *****/
1402      void
1403      EDRessubmitElement::setSocketHost(char *sockethost)
1404      {
1405          if (sockethost != NULL)
1406          {
1407              if (se_socket_host != NULL)
1408              {
1409                  free(se_socket_host);
1410              }
1411              se_socket_host = strdup(sockethost);
1412          }
1413      }
1414      /*****
1415
1416      ** Routine: setScriptName
1417      **
1418      ** Inputs:  char *scriptname - script to run on the client
1419      **
1420      ** Outputs: None
1421      **
1422      ** Return Codes:
1423      **      None
1424      **
1425      ** Purpose: Sets the scriptname to be used on the client using the
1426      **      specified buffer.
1427      *****/
1428      void
1429      EDRessubmitElement::setScriptName(char *scriptname)
1430      {
1431          if (scriptname != NULL)
1432          {
1433              if (se_client_scriptname != NULL)
1434              {
1435                  free(se_client_scriptname);
1436              }
1437              se_client_scriptname = strdup(scriptname);
1438          }
1439      }
1440      /*****
1441
1442      ** Routine: setSocketHost
1443      **
1444      ** Inputs:  char *sockethost - hostname to connect to
1445      **
1446      ** Outputs: None
1447      **
1448      ** Return Codes:
1449      **      None
1450      **
1451      ** Purpose: Sets the hostname to connect to using the specified
1452      **      buffer.
1453      *****/
1454      void
1455      EDRessubmitElement::setSocketHost(char *sockethost)
1456      {
1457          if (sockethost != NULL)
1458          {
1459              if (se_socket_host != NULL)
1460              {
1461                  free(se_socket_host);
1462              }
1463              se_socket_host = strdup(sockethost);
1464          }
1465      }
1466      /*****
1467
1468      ** Routine: setScriptName
1469      **
1470      ** Inputs:  char *scriptname - script to run on the client
1471      **
1472      ** Outputs: None
1473      **
1474      ** Return Codes:
1475      **      None
1476      **
1477      ** Purpose: Sets the scriptname to be used on the client using the
1478      **      specified buffer.
1479      *****/
1480      void
1481      EDRessubmitElement::setScriptName(char *scriptname)
1482      {
1483          if (scriptname != NULL)
1484          {
1485              if (se_client_scriptname != NULL)
1486              {
1487                  free(se_client_scriptname);
1488              }
1489              se_client_scriptname = strdup(scriptname);
1490          }
1491      }
1492      /*****
1493
1494      ** Routine: setSocketHost
1495      **
1496      ** Inputs:  char *sockethost - hostname to connect to
1497      **
1498      ** Outputs: None
1499      **
1500      ** Return Codes:
1501      **      None
1502      **
1503      ** Purpose: Sets the hostname to connect to using the specified
1504      **      buffer.
1505      *****/
1506      void
1507      EDRessubmitElement::setSocketHost(char *sockethost)
1508      {
1509          if (sockethost != NULL)
1510          {
1511              if (se_socket_host != NULL)
1512              {
1513                  free(se_socket_host);
1514              }
1515              se_socket_host = strdup(sockethost);
1516          }
1517      }
1518      /*****
1519
1520      ** Routine: setScriptName
1521      **
1522      ** Inputs:  char *scriptname - script to run on the client
1523      **
1524      ** Outputs: None
1525      **
1526      ** Return Codes:
1527      **      None
1528      **
1529      ** Purpose: Sets the scriptname to be used on the client using the
1530      **      specified buffer.
1531      *****/
1532      void
1533      EDRessubmitElement::setScriptName(char *scriptname)
1534      {
1535          if (scriptname != NULL)
1536          {
1537              if (se_client_scriptname != NULL)
1538              {
1539                  free(se_client_scriptname);
1540              }
1541              se_client_scriptname = strdup(scriptname);
1542          }
1543      }
1544      /*****
1545
1546      ** Routine: setSocketHost
1547      **
1548      ** Inputs:  char *sockethost - hostname to connect to
1549      **
1550      ** Outputs: None
1551      **
1552      ** Return Codes:
1553      **      None
1554      **
1555      ** Purpose: Sets the hostname to connect to using the specified
1556      **      buffer.
1557      *****/
1558      void
1559      EDRessubmitElement::setSocketHost(char *sockethost)
1560      {
1561          if (sockethost != NULL)
1562          {
1563              if (se_socket_host != NULL)
1564              {
1565                  free(se_socket_host);
1566              }
1567              se_socket_host = strdup(sockethost);
1568          }
1569      }
1570      /*****
1571
1572      ** Routine: setScriptName
1573      **
1574      ** Inputs:  char *scriptname - script to run on the client
1575      **
1576      ** Outputs: None
1577      **
1578      ** Return Codes:
1579      **      None
1580      **
1581      ** Purpose: Sets the scriptname to be used on the client using the
1582      **      specified buffer.
1583      *****/
1584      void
1585      EDRessubmitElement::setScriptName(char *scriptname)
1586      {
1587          if (scriptname != NULL)
1588          {
1589              if (se_client_scriptname != NULL)
1590              {
1591                  free(se_client_scriptname);
1592              }
1593              se_client_scriptname = strdup(scriptname);
1594          }
1595      }
1596      /*****
1597
1598      ** Routine: setSocketHost
1599      **
1600      ** Inputs:  char *sockethost - hostname to connect to
1601      **
1602      ** Outputs: None
1603      **
1604      ** Return Codes:
1605      **      None
1606      **
1607      ** Purpose: Sets the hostname to connect to using the specified
1608      **      buffer.
1609      *****/
1610      void
1611      EDRessubmitElement::setSocketHost(char *sockethost)
1612      {
1613          if (sockethost != NULL)
1614          {
1615              if (se_socket_host != NULL)
1616              {
1617                  free(se_socket_host);
1618              }
1619              se_socket_host = strdup(sockethost);
1620          }
1621      }
1622      /*****
1623
1624      ** Routine: setScriptName
1625      **
1626      ** Inputs:  char *scriptname - script to run on the client
1627      **
1628      ** Outputs: None
1629      **
1630      ** Return Codes:
1631      **      None
1632      **
1633      ** Purpose: Sets the scriptname to be used on the client using the
1634      **      specified buffer.
1635      *****/
1636      void
1637      EDRessubmitElement::setScriptName(char *scriptname)
1638      {
1639          if (scriptname != NULL)
1640          {
1641              if (se_client_scriptname != NULL)
1642              {
1643                  free(se_client_scriptname);
1644              }
1645              se_client_scriptname = strdup(scriptname);
1646          }
1647      }
1648      /*****
1649
1650      ** Routine: setSocketHost
1651      **
1652      ** Inputs:  char *sockethost - hostname to connect to
1653      **
1654      ** Outputs: None
1655      **
1656      ** Return Codes:
1657      **      None
1658      **
1659      ** Purpose: Sets the hostname to connect to using the specified
1660      **      buffer.
1661      *****/
1662      void
1663      EDRessubmitElement::setSocketHost(char *sockethost)
1664      {
1665          if (sockethost != NULL)
1666          {
1667              if (se_socket_host != NULL)
1668              {
1669                  free(se_socket_host);
1670              }
1671              se_socket_host = strdup(sockethost);
1672          }
1673      }
1674      /*****
1675
1676      ** Routine: setScriptName
1677      **
1678      ** Inputs:  char *scriptname - script to run on the client
1679      **
1680      ** Outputs: None
1681      **
1682      ** Return Codes:
1683      **      None
1684      **
1685      ** Purpose: Sets the scriptname to be used on the client using the
1686      **      specified buffer.
1687      *****/
1688      void
1689      EDRessubmitElement::setScriptName(char *scriptname)
1690      {
1691          if (scriptname != NULL)
1692          {
1693              if (se_client_scriptname != NULL)
1694              {
1695                  free(se_client_scriptname);
1696              }
1697              se_client_scriptname = strdup(scriptname);
1698          }
1699      }
1700      /*****
1701
1702      ** Routine: setSocketHost
1703      **
1704      ** Inputs:  char *sockethost - hostname to connect to
1705      **
1706      ** Outputs: None
1707      **
1708      ** Return Codes:
1709      **      None
1710      **
1711      ** Purpose: Sets the hostname to connect to using the specified
1712      **      buffer.
1713      *****/
1714      void
1715      EDRessubmitElement::setSocketHost(char *sockethost)
1716      {
1717          if (sockethost != NULL)
1718          {
1719              if (se_socket_host != NULL)
1720              {
1721                  free(se_socket_host);
1722              }
1723              se_socket_host = strdup(sockethost);
1724          }
1725      }
1726      /*****
1727
1728      ** Routine: setScriptName
1729      **
1730      ** Inputs:  char *scriptname - script to run on the client
1731      **
1732      ** Outputs: None
1733      **
1734      ** Return Codes:
1735      **      None
1736      **
1737      ** Purpose: Sets the scriptname to be used on the client using the
1738      **      specified buffer.
1739      *****/
1740      void
1741      EDRessubmitElement::setScriptName(char *scriptname)
1742      {
1743          if (scriptname != NULL)
1744          {
1745              if (se_client_scriptname != NULL)
1746              {
1747                  free(se_client_scriptname);
1748              }
1749              se_client_scriptname = strdup(scriptname);
1750          }
1751      }
1752      /*****
1753
1754      ** Routine: setSocketHost
1755      **
1756      ** Inputs:  char *sockethost - hostname to connect to
1757      **
1758      ** Outputs: None
1759      **
1760      ** Return Codes:
1761      **      None
1762      **
1763      ** Purpose: Sets the hostname to connect to using the specified
1764      **      buffer.
1765      *****/
1766      void
1767      EDRessubmitElement::setSocketHost(char *sockethost)
1768      {
1769          if (sockethost != NULL)
1770          {
1771              if (se_socket_host != NULL)
1772              {
1773                  free(se_socket_host);
1774              }
1775              se_socket_host = strdup(sockethost);
1776          }
1777      }
1778      /*****
1779
1780      ** Routine: setScriptName
1781      **
1782      ** Inputs:  char *scriptname - script to run on the client
1783      **
1784      ** Outputs: None
1785      **
1786      ** Return Codes:
1787      **      None
1788      **
1789      ** Purpose: Sets the scriptname to be used on the client using the
1790      **      specified buffer.
1791      *****/
1792      void
1793      EDRessubmitElement::setScriptName(char *scriptname)
1794      {
1795          if (scriptname != NULL)
1796          {
1797              if (se_client_scriptname != NULL)
1798              {
1799                  free(se_client_scriptname);
1800              }
1801              se_client_scriptname = strdup(scriptname);
1802          }
1803      }
1804      /*****
1805
1806      ** Routine: setSocketHost
1807      **
1808      ** Inputs:  char *sockethost - hostname to connect to
1809      **
1810      ** Outputs: None
1811      **
1812      ** Return Codes:
1813      **      None
1814      **
1815      ** Purpose: Sets the hostname to connect to using the specified
1816      **      buffer.
1817      *****/
1818      void
1819      EDRessubmitElement::setSocketHost(char *sockethost)
1820      {
1821          if (sockethost != NULL)
1822          {
1823              if (se_socket_host != NULL)
1824              {
1825                  free(se_socket_host);
1826              }
1827              se_socket_host = strdup(sockethost);
1828          }
1829      }
1830      /*****
1831
1832      ** Routine: setScriptName
1833      **
1834      ** Inputs:  char *scriptname - script to run on the client
1835      **
1836      ** Outputs: None
1837      **
1838      ** Return Codes:
1839      **      None
1840      **
1841      ** Purpose: Sets the scriptname to be used on the client using the
1842      **      specified buffer.
1843      *****/
1844      void
1845      EDRessubmitElement::setScriptName(char *scriptname)
1846      {
1847          if (scriptname != NULL)
1848          {
1849              if (se_client_scriptname != NULL)
1850              {
1851                  free(se_client_scriptname);
1852              }
1853              se_client_scriptname = strdup(scriptname);
1854          }
1855      }
1856      /*****
1857
1858      ** Routine: setSocketHost
1859      **
1860      ** Inputs:  char *sockethost - hostname to connect to
1861      **
1862      ** Outputs: None
1863      **
1864      ** Return Codes:
1865      **      None
1866      **
1867      ** Purpose: Sets the hostname to connect to using the specified
1868      **      buffer.
1869      *****/
1870      void
1871      EDRessubmitElement::setSocketHost(char *sockethost)
1872      {
1873          if (sockethost != NULL)
1874          {
1875              if (se_socket_host != NULL)
1876              {
1877                  free(se_socket_host);
1878              }
1879              se_socket_host = strdup(sockethost);
1880          }
1881      }
1882      /*****
1883
1884      ** Routine: setScriptName
1885      **
1886      ** Inputs:  char *scriptname - script to run on the client
1887      **
1888      ** Outputs: None
1889      **
1890      ** Return Codes:
1891      **      None
1892      **
1893      ** Purpose: Sets the scriptname to be used on the client using the
1894      **      specified buffer.
1895      *****/
1896      void
1897      EDRessubmitElement::setScriptName(char *scriptname)
1898      {
1899          if (scriptname != NULL)
1900          {
1901              if (se_client_scriptname != NULL)
1902              {
1903                  free(se_client_scriptname);
1904              }
1905              se_client_scriptname = strdup(scriptname);
1906          }
1907      }
1908      /*****
1909
1910      ** Routine: setSocketHost
1911      **
1912      ** Inputs:  char *sockethost - hostname to connect to
1913      **
1914      ** Outputs: None
1915      **
1916      ** Return Codes:
1917      **      None
1918      **
1919      ** Purpose: Sets the hostname to connect to using the specified
1920      **      buffer.
1921      *****/
1922      void
1923      EDRessubmitElement::setSocketHost(char *sockethost)
1924      {
1925          if (sockethost != NULL)
1926          {
1927              if (se_socket_host != NULL)
1928              {
1929                  free(se_socket_host);
1930              }
1931              se_socket_host = strdup(sockethost);
1932          }
1933      }
1934      /*****
1935
1936      ** Routine: setScriptName
1937      **
1938      ** Inputs:  char *scriptname - script to run on the client
1939      **
1940      ** Outputs: None
1941      **
1942      ** Return Codes:
1943      **      None
1944      **
1945      ** Purpose: Sets the scriptname to be used on the client using the
1946      **      specified buffer.
1947      *****/
1948      void
1949      EDRessubmitElement::setScriptName(char *scriptname)
1950      {
1951          if (scriptname != NULL)
1952          {
1953              if (se_client_scriptname != NULL)
1954              {
1955                  free(se_client_scriptname);
1956              }
1957              se_client_scriptname = strdup(scriptname);
1958          }
1959      }
1960      /*****
1961
1962      ** Routine: setSocketHost
1963      **
1964      ** Inputs:  char *sockethost - hostname to connect to
1965      **
1966      ** Outputs: None
1967      **
1968      ** Return Codes:
1969      **      None
1970      **
1971      ** Purpose: Sets the hostname to connect to using the specified
1972      **      buffer.
1973      *****/
1974      void
1975      EDRessubmitElement::setSocketHost(char *sockethost)
1976      {
1977          if (sockethost != NULL)
1978          {
1979              if (se_socket_host != NULL)
1980              {
1981                  free(se_socket_host);
1982              }
1983              se_socket_host = strdup(sockethost);
1984          }
1985      }
1986      /*****
1987
1988      ** Routine: setScriptName
1989      **
1990      ** Inputs:  char *scriptname - script to run on the client
1991      **
1992      ** Outputs: None
1993      **
1994      ** Return Codes:
1995      **      None
1996      **
1997      ** Purpose: Sets the scriptname to be used on the client using the
1998      **      specified buffer.
1999      *****/
2000      void
2001      EDRessubmitElement::setScriptName(char *scriptname)
2002      {
2003          if (scriptname != NULL)
2004          {
2005              if (se_client_scriptname != NULL)
2006              {
2007                  free(se_client_scriptname);
2008              }
2009              se_client_scriptname = strdup(scriptname);
2010          }
2011      }
2012      /*****
2013
2014      ** Routine: setSocketHost
2015      **
2016      ** Inputs:  char *sockethost - hostname to connect to
2017      **
2018      ** Outputs: None
2019      **
2020      ** Return Codes:
2021      **      None
2022      **
2023      ** Purpose: Sets the hostname to connect to using the specified
2024      **      buffer.
2025      *****/
2026      void
2027      EDRessubmitElement::setSocketHost(char *sockethost)
2028      {
2029          if (sockethost != NULL)
2030          {
2031              if (se_socket_host != NULL)
2032              {
2033                  free(se_socket_host);
2034              }
2035              se_socket_host = strdup(sockethost);
2036          }
2037      }
2038      /*****
2039
2040      ** Routine: setScriptName
2041      **
2042      ** Inputs:  char *scriptname - script to run on the client
2043      **
2044      ** Outputs: None
2045      **
2046      ** Return Codes:
2047      **      None
2048      **
2049      ** Purpose: Sets the scriptname to be used on the client using the
2050      **      specified buffer.
2051      *****/
2052      void
2053      EDRessubmitElement::setScriptName(char *scriptname)
2054      {
2055          if (scriptname != NULL)
2056          {
2057              if (se_client_scriptname != NULL)
2058              {
2059                  free(se_client_scriptname);
2060              }
2061              se_client_scriptname = strdup(scriptname);
2062          }
2063      }
2064      /*****
2065
2066      ** Routine: setSocketHost
2067      **
2068      ** Inputs:  char *sockethost - hostname to connect to
2069      **
2070      ** Outputs: None
2071      **
2072      ** Return Codes:
2073      **      None
2074      **
2075      ** Purpose: Sets the hostname to connect to using the specified
2076      **      buffer.
2077      *****/
2078      void
2079      EDRessubmitElement::setSocketHost(char *sockethost)
2080      {
2081          if (sockethost != NULL)
2082          {
2083              if (se_socket_host != NULL)
2084              {
2085                  free(se_socket_host);
2086              }
2087              se_socket_host = strdup(sockethost);
2088          }
2089      }
2090      /*****
2091
2092      ** Routine: setScriptName
2093      **
2094      ** Inputs:  char *scriptname - script to run on the client
2095      **
2096      ** Outputs: None
2097      **
2098      ** Return Codes:
2099      **      None
2100      **
2101      ** Purpose: Sets the scriptname to be used on the client using the
2102      **      specified buffer.
2103      *****/
2104      void
2105      EDRessubmitElement::setScriptName(char *scriptname)
2106      {
2107          if (scriptname != NULL)
2108          {
2109              if (se_client_scriptname != NULL)
2110              {
2111                  free(se_client_scriptname);
2112              }
2113              se_client_scriptname = strdup(scriptname);
2114          }
2115      }
2116      /*****
2117
2118      ** Routine: setSocketHost
2119      **
2120      ** Inputs:  char *sockethost - hostname to connect to
2121      **
2122      ** Outputs: None
2123      **
2124      ** Return Codes:
2125      **      None
2126      **
2127      ** Purpose: Sets the hostname to connect to using the specified
2128      **      buffer.
2129      *****/
2130      void
2131      EDRessubmitElement::setSocketHost(char *sockethost)
2132      {
2133          if (sockethost != NULL)
2134          {
2135              if (se_socket_host != NULL)
2136              {
2137                  free(se_socket_host);
2138              }
2139              se_socket_host = strdup(sockethost);
2140          }
2141      }
2142      /*****
2143
2144      ** Routine: setScriptName
2145      **
2146      ** Inputs:  char *scriptname - script to run on the client
2147      **
2148      ** Outputs: None
2149      **
2150      ** Return Codes:
2151      **      None
2152      **
2153      ** Purpose: Sets the scriptname to be used on the client using the
2154      **      specified buffer.
2155      *****/
2156      void
2157      EDRessubmitElement::setScriptName(char *scriptname)
2158      {
2159          if (scriptname != NULL)
2160          {
2161              if (se_client_scriptname != NULL)
2162              {
2163                  free(se_client_scriptname);
2164              }
2165              se_client_scriptname = strdup(scriptname);
2166          }
2167      }
2168      /*****
2169
2170      ** Routine: setSocketHost
2171      **
2172      ** Inputs:  char *sockethost - hostname to connect to
2173      **
2174      ** Outputs: None
2175      **
2176      ** Return Codes:
2177      **      None
2178      **
2179      ** Purpose: Sets the hostname to connect to using the specified
2180      **      buffer.
2181      *****/
2182      void
2183      EDRessubmitElement::
```



```

1218 */
1219
1220 void
1221 EDRESsubmitElement::getVolumeList(ebv_volldlist_ty **volptr)
1222 {
1223     if (volptr != NULL)
1224         *volptr = se_work_item_volume_list;
1225 }
1226
1227 /*****
1228 **
1229 ** Routine: setVolumeList
1230
1231 ** Inputs: ebv_volldlist_ty volptr - place to put the volume list
1232
1233 ** Outputs: None
1234
1235 ** Return Codes: None
1236
1237 ** Purpose: Sets the volume list using the specified buffer.
1238
1239 **
1240 *****/
1241
1242 void
1243 EDRESsubmitElement::setVolumeList(ebv_volldlist_ty *volptr)
1244 {
1245     if (volptr != NULL)
1246         se_work_item_volume_list = volptr;
1247 }
1248
1249 /*****
1250 **
1251 ** Routine: getSubmitFile
1252
1253 ** Inputs: int maxlen - the size of the buffer passed in
1254
1255 ** Outputs: char *submitfile - place to put the submit file name
1256
1257 ** Return Codes: None
1258
1259 ** Purpose: Returns the submit file name in the specified buffer.
1260
1261 **
1262 *****/
1263
1264 char *
1265 EDRESsubmitElement::getSubmitFile(char *submitfile, int maxlen)
1266 {
1267     if (submitfile != NULL && se_submit_file_name != NULL)
1268     {
1269         strncpy(submitfile, se_submit_file_name, maxlen);
1270         submitfile[maxlen - 1] = 0;
1271     }
1272     else if (submitfile != NULL)
1273     {
1274         submitfile[0] = 0;
1275     }
1276 }
1277
1278 }
1279
1280

```

```

1280
1281 /*****
1282 **
1283 ** Routine: setSubmitFile
1284
1285 ** Inputs: None
1286
1287 ** Outputs: int *status - return the status of this operation,
1288                the errno.
1289
1290 ** Return Codes: None
1291
1292 ** Purpose: Sets the submit file name and makes sure it is unique.
1293
1294 *****/
1295
1296 int
1297 EDRESsubmitElement::setSubmitFile(char *submitfile, int *status)
1298 {
1299     if (NULL != submitfile)
1300     {
1301         if (NULL != se_submit_file_name)
1302         {
1303             free(se_submit_file_name);
1304         }
1305         se_submit_file_name = strdup(submitfile);
1306         if (NULL == se_submit_file_name)
1307         {
1308             *status = ENOMEM;
1309         }
1310     }
1311     else
1312     {
1313         *status = EINVAL;
1314     }
1315     return;
1316 }
1317
1318 /*****
1319 **
1320 ** Routine: getSocketPort
1321
1322 ** Inputs: None
1323
1324 ** Outputs: None
1325
1326 ** Return Codes: char - the work item type
1327
1328 ** Purpose: Returns the work item type
1329
1330 **
1331 *****/
1332
1333 char
1334 EDRESsubmitElement::getWorkItemType()
1335 {
1336     return w_i_type;
1337 }
1338
1339

```


1	/*	55	*****	*****
2	/* Copyright 1996,1997 EMC Corporation	56	*****	*****
3	*/	67	*/	*****
4		68		*****
5	/* EDMRESubmitObj.cc	69	EDMRESubmitObj::EDMRESubmitObj()	
6		70	{	
7		71	submitId = 0;	
8		72	so.submit_type = 0;	
9	Mission Statement: file that contains the Handle class methods	73	so.execute_flag = 0;	
10		74	so.no_vm_check = FALSE;	
11	Primary Data Acct On:	75	pre_phase_executable = NULL;	
12		76	post_phase_executable = NULL;	
13	Compile-Time Options:	77	pre_phase_env = NULL;	
14		78	post_phase_env = NULL;	
15	Basic Idea here:	79		
16		80	execute_override_phase_executable = NULL;	
17	The handle object is a container which holds a	81	execute_override_phase_argv = NULL;	
18	set of handles for each running auxproc.	82	execute_override_phase_env = NULL;	
19	*/			
20	#if defined(lint)	83	post_phase_executable = NULL;	
21	static char RCS_id [] = "%(\$SRCFILE: EDMRESubmitObj.cc,v \$ "	84	post_phase_argv = NULL;	
22		85	post_phase_env = NULL;	
23	\$Revision: 1.0 \$ "	86	post_phase_argv = NULL;	
24	\$Date: 1997/02/06 20:49:15 \$ "			
25	#endif	87	so.backup administrator = FALSE;	
26	#include <osl/c_portable.h>	88	so.svc_sys_admin = FALSE;	
27	#include <osl/ep_xopen.h>	89	so.dtl_sys_admin = FALSE;	
28	#include <osl/inout.h>	90		
29		91		
30	#include <string.h>	92	so.human_uid = -1;	
31	#include <stdlib.h>	93	so.human_username = NULL;	
32	// Require Name, includes	94	so.effective_uid = -1;	
33	#include <cw/collct.h>	95	so.effective_username = NULL;	
34	#include <cw/cfile.h>	96	memset(&so.local_submit_summary, 0, sizeof(struct mark_summary));	
35	#include <cw/vstream.h>	97	so.local_volume_list = NULL;	
36		98	so.witem_count = 0;	
37	#include <cw/bintree.h>	99		
38		100	}	
39	#include <restore/RestoreObjectID.h>	101	/*	*****
40	#include <restore/restore/dispatch_daemon.h>	102	/*	*****
41	#include <restore/restore/api.h>	103	Routine: EDMRESubmitObj destructor	
42	#include <restore/RestoreContext.h>	104		
43	#include <osreport/objv1.h>	105	Inputs: None	
44		106	Outputs: None	
45	#include <restore/EDMRESubmitElement.h>	107	Return Codes: None	
46	#include <restore/EDMRESubmitObj.h>	108		
47		109	Purpose: Doesn't really do anything but seems to be a requirement	
48	// Needed for objects wave linked list manager.	110	for the linked list manager.	
49	// 408 is the object ID.	111	*****	
50	RMDEFINTE_COLLSTRUCTABLE(EDMRESubmitObj, EDMRESUBMITOBJ)	112	*****	
51	RMDEFINTE_COLLSTRUCTABLE(EDMRESubmitObj, EDMRESUBMITOBJ)	113	*****	
52	RMDEFINTE_COLLSTRUCTABLE(EDMRESubmitObj, EDMRESUBMITOBJ)	114	*****	
53	RMDEFINTE_COLLSTRUCTABLE(EDMRESubmitObj, EDMRESUBMITOBJ)	115	*****	
54	RMDEFINTE_COLLSTRUCTABLE(EDMRESubmitObj, EDMRESUBMITOBJ)	116	*****	
55	RMDEFINTE_COLLSTRUCTABLE(EDMRESubmitObj, EDMRESUBMITOBJ)	117	*****	
56	RMDEFINTE_COLLSTRUCTABLE(EDMRESubmitObj, EDMRESUBMITOBJ)	118	*****	
57	RMDEFINTE_COLLSTRUCTABLE(EDMRESubmitObj, EDMRESUBMITOBJ)	119	*****	
58	RMDEFINTE_COLLSTRUCTABLE(EDMRESubmitObj, EDMRESUBMITOBJ)	120	*****	
59	RMDEFINTE_COLLSTRUCTABLE(EDMRESubmitObj, EDMRESUBMITOBJ)	121	*****	
60	RMDEFINTE_COLLSTRUCTABLE(EDMRESubmitObj, EDMRESUBMITOBJ)	122	*****	
61	RMDEFINTE_COLLSTRUCTABLE(EDMRESubmitObj, EDMRESUBMITOBJ)	123	*****	
62	RMDEFINTE_COLLSTRUCTABLE(EDMRESubmitObj, EDMRESUBMITOBJ)	124	*****	
63	RMDEFINTE_COLLSTRUCTABLE(EDMRESubmitObj, EDMRESUBMITOBJ)	125	*****	
64	RMDEFINTE_COLLSTRUCTABLE(EDMRESubmitObj, EDMRESUBMITOBJ)	126	*****	
65	RMDEFINTE_COLLSTRUCTABLE(EDMRESubmitObj, EDMRESUBMITOBJ)	127	*****	
66	RMDEFINTE_COLLSTRUCTABLE(EDMRESubmitObj, EDMRESUBMITOBJ)	128	*****	
67	RMDEFINTE_COLLSTRUCTABLE(EDMRESubmitObj, EDMRESUBMITOBJ)	129	*****	
68	RMDEFINTE_COLLSTRUCTABLE(EDMRESubmitObj, EDMRESUBMITOBJ)	130	*****	
69	RMDEFINTE_COLLSTRUCTABLE(EDMRESubmitObj, EDMRESUBMITOBJ)	131	*****	
70	RMDEFINTE_COLLSTRUCTABLE(EDMRESubmitObj, EDMRESUBMITOBJ)	132	*****	
71	RMDEFINTE_COLLSTRUCTABLE(EDMRESubmitObj, EDMRESUBMITOBJ)	133	*****	
72	RMDEFINTE_COLLSTRUCTABLE(EDMRESubmitObj, EDMRESUBMITOBJ)	134	*****	
73	RMDEFINTE_COLLSTRUCTABLE(EDMRESubmitObj, EDMRESUBMITOBJ)	135	*****	
74	RMDEFINTE_COLLSTRUCTABLE(EDMRESubmitObj, EDMRESUBMITOBJ)	136	*****	
75	RMDEFINTE_COLLSTRUCTABLE(EDMRESubmitObj, EDMRESUBMITOBJ)	137	*****	
76	RMDEFINTE_COLLSTRUCTABLE(EDMRESubmitObj, EDMRESUBMITOBJ)	138	*****	
77	RMDEFINTE_COLLSTRUCTABLE(EDMRESubmitObj, EDMRESUBMITOBJ)	139	*****	
78	RMDEFINTE_COLLSTRUCTABLE(EDMRESubmitObj, EDMRESUBMITOBJ)	140	*****	
79	RMDEFINTE_COLLSTRUCTABLE(EDMRESubmitObj, EDMRESUBMITOBJ)	141	*****	
80	RMDEFINTE_COLLSTRUCTABLE(EDMRESubmitObj, EDMRESUBMITOBJ)	142	*****	

Page 223 of 248		
128 1	if (post_phase_executable != NULL)	
129 1	free(post_phase_executable);	
130 1	if (so_human_uidname != NULL)	
131 1	free(so_human_uidname);	
132 1		
133 1	if (so_effective_uidname != NULL)	
134 1	free(so_effective_uidname);	
135 1		
136 1	/*	
137 1		
138 1	Currently we aren't managing/creating this memory so don't free it.	
139 1		
140 1	if (pre_phase_argv != NULL)	
141 1	{	
142 1	int i = 0;	
143 1	while (pre_phase_argv[i] != NULL)	
144 1	free(pre_phase_argv[i++]);	
145 1		
146 1	free(pre_phase_argv);	
147 1	}	
148 1		
149 1	if (pre_phase_argv != NULL)	
150 1	{	
151 1	int i = 0;	
152 1	while (pre_phase_argv[i] != NULL)	
153 1	free(pre_phase_argv[i++]);	
154 1		
155 1	free(pre_phase_argv);	
156 1	}	
157 1		
158 1	if (execute_override_phase_argv != NULL)	
159 1	{	
160 1	int i = 0;	
161 1	while (execute_override_phase_argv[i] != NULL)	
162 1	free (execute_override_phase_argv[i++]);	
163 1		
164 1	free (execute_override_phase_argv);	
165 1	}	
166 1		
167 1	if (execute_override_phase_argv != NULL)	
168 1	{	
169 1	int i = 0;	
170 1	while (execute_override_phase_argv[i] != NULL)	
171 1	free (execute_override_phase_argv[i++]);	
172 1		
173 1	free (execute_override_phase_argv);	
174 1	}	
175 1		
176 1	if (post_phase_argv != NULL)	
177 1	{	
178 1	int i = 0;	
179 1	while (post_phase_argv[i] != NULL)	
180 1	free (post_phase_argv[i++]);	
181 1		
182 1	free (post_phase_argv);	
183 1	}	
184 1		
185 1	if (post_phase_argv != NULL)	
186 1	{	
187 1	int i = 0;	
188 1	while (post_phase_argv[i] != NULL)	
189 1	free (post_phase_argv[i++]);	
190 1		
191 1	free (post_phase_argv);	
192 1	}	
193 1		
Page 223 of 248	./libo_restore/EDMResturnObj.cc 3	Fri Jan 04 16:35:25 2008

Page 224 of 248		
194 1	/*	
195 1	// Need to free mark summary and volume summary here.	
196 1	}	
197 1		
198 1	/*	
199 1		
200 1	ROUTINE: compareto	
201 1	/*	
202 1	Inputs: RMCollectable *c - a pointer to the base class type which	
203 1	you can then cast and compare.	
204 1	/*	
205 1	Outputs: None	
206 1	/*	
207 1	Return Codes: - returns numbers like groc compare (-1, 0, 1)	
208 1	int	
209 1	/*	
210 1	Purpose: Compare using the submit ID.	
211 1	/*	
212 1		
213 1		
214 1	/*	
215 1	EDMResturnObj::compareto (IN const RMCollectable *c) const	
216 1	{	
217 1	EDMResturnObj *localcmd = (EDMResturnObj *) c;	
218 1		
219 1	if (localcmd == NULL)	
220 1	return -1;	
221 1		
222 1	if (localcmd == NULL)	
223 1	return -1;	
224 1	if (localcmd -> submitID > submitID)	
225 1	return 1;	
226 1	else if (localcmd -> submitID < submitID)	
227 1	return -1;	
228 1		
229 1	return 0;	
230 1	}	
231 1	/*	
232 1		
233 1	ROUTINE: isequal	
234 1	/*	
235 1	Inputs: RMCollectable *c - a pointer to the base class type which	
236 1	you can then cast and compare.	
237 1	/*	
238 1	Outputs: None	
239 1	/*	
240 1	Return Codes: RMBoolean - TRUE or FALSE	
241 1	/*	
242 1	Purpose: Compare submit IDs to see if it is the same submit object.	
243 1	/*	
244 1		
245 1		
246 1		
247 1	/*	
248 1	RMBoolean	
249 1	EDMResturnObj::isequal (IN const RMCollectable *c) const	
250 1	{	
251 1	EDMResturnObj *localcmd = (EDMResturnObj *) c;	
252 1	if (localcmd == NULL)	
253 1	return FALSE;	
254 1		
255 1		
Page 224 of 248	./libo_restore/EDMResturnObj.cc 4	Fri Jan 04 16:35:25 2008

```

257 1      if (!localCmd -> submitID == submitID)
258 1          return TRUE;
259 1
260 1      return FALSE;
261 1    }
262 1
263 1    /*****
264 1     **
265 1     ** Routine: hash
266 1     **
267 1     ** Inputs: None
268 1     **
269 1     ** Outputs: None
270 1     **
271 1     ** Return Codes:
272 1         unsigned - returns submit ID.
273 1     **
274 1     ** Purpose: Returns unique value, in this case submit ID.
275 1     **
276 1     *****/
277 1
278 1    unsigned
279 1    EDMSRESumObj::hash() const
280 1    {
281 1        return (unsigned) submitID;
282 1    }
283 1
284 1    /*****/
285 1
286 1    **
287 1    ** Routine: saveOutputs
288 1    **
289 1    ** Inputs: RWFile f - file pointer where data will be saved.
290 1    **
291 1    ** Outputs: None
292 1    **
293 1    ** Return Codes:
294 1        None
295 1    **
296 1    ** Purpose: Save class internal data to a file.
297 1    **
298 1    *****/
299 1
300 1    void
301 1    EDMSRESumObj::saveData(IN RWFile &f)
302 1    {
303 1        // Save parent class data too
304 1        RMCollactable::saveOutputs(f);
305 1        // Left as an exercise
306 1    }
307 1
308 1    /*****/
309 1
310 1    /*****/
311 1
312 1    ** Routine: saveOutputs
313 1    **
314 1    ** Inputs: RMCollactable stream - stream to write internal data to.
315 1    **
316 1    ** Outputs: None

```

```

3118      ** Return Codes:
3119      None
3120      **
3121      ** Purpose: Save class data to a stream.
3122      **
3123      ..
3124      */
3125      void
3126      EDMSRlimitIOJ::saveOutputs(IN RWostream &strm)
3127      {
3128          // Save parent class data too
3129          RMCollectable::saveOutputs(strm);
3130      }
3131      // Left as an exercise
3132      ..
3133      ..
3134      ..
3135      ..
3136      ..
3137      ..
3138      ..
3139      ..
3140      ..
3141      ..
3142      ..
3143      ..
3144      ..
3145      ..
3146      ..
3147      ..
3148      ..
3149      ..
3150      ..
3151      ..
3152      ..
3153      ..
3154      ..
3155      ..
3156      ..
3157      ..
3158      ..
3159      ..
3160      ..
3161      ..
3162      ..
3163      ..
3164      ..
3165      ..
3166      ..
3167      ..
3168      ..
3169      ..
3170      ..
3171      ..
3172      ..
3173      ..
3174      ..
3175      ..

```

```

376 */
379 void
379 EDRESsubmitObj::restoreData(IN RWtistream& strm)
380 { // Restore parent data too
381     RWcollecible::restoreData(strm);
382 }
383 // Left as an exercise
384
385 }
386
387 /*****
388 **
388 ** Routine: binaryScoresize
389 **
390 ** Inputs: None
391 **
391 ** Outputs: None
392
393
394 ** Return Codes:
395 **
395 ** Rhspace count - file size of class written to disk in
396 bytes
397
397 ** Purpose: Returns the size of class if it were stored on disk.
398
399 *****/
400
401 */
402
403 Rhspace
403 EDRESsubmitObj::binaryScoresize() const
404 {
404     Rhspace count = RWcollecible::binaryScoresize() +
405         sizeof(submitID);
406     // Left as an exercise
407     return count;
408 }
409
410
411 /*****
412 **
412 ** Routine: deleteSubmitFiles
413 **
413 ** Inputs: None
414 **
414 ** Outputs: None
415
416
417 ** Return Codes:
418 **
418 ** Int - the submit ID
419
420 ** Purpose: Returns the submit ID.
421
422 *****/
423
424
425 //
426 EDRESsubmitObj::deleteSubmitFiles()
427 {
428     // The iterator to traverse the tree of submit elements
429     RWBinaryTreeIterator<submitElement> elem;
430     EDRESsubmitElement *elem;
431     char *submitfile; // the name of the submit file to delete
432     int maxlen = MAXPATHLEN; // the max size of the submitfile
433     submitfile = (char *) calloc(maxlen,
434         // callloc enough memory for the submit file
435         1);
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498

```

```

416 1 // if the submit file is null, we have a callloc failure
417 1 {
418 1     if (NULL == submitfile)
419 1     {
420 1         return -1;
421 1     }
422 1
423 1 // create an iterator to traverse the tree
424 1 submitElementTree elem(RWBinaryTreeIterator(submitElements));
425 1 if (NULL == submitElementTree)
426 1     {
427 1         free(submitfile);
428 1         return (-1);
429 1     }
430 1 //reset the iterator
431 1 submitElementTree->reset();
432 1 elem = submitElementTree->begin() and get the
433 1 // submitfile name and unlink it
434 1 while(NULL != (*submitElementTree)())
435 1 {
436 1     elem = (*submitElementTree->key());
437 1     if (NULL == elem)
438 1     {
439 1         free(submitfile);
440 1         return -1;
441 1     }
442 1 elem->getSubmitFile(submitfile, maxlen);
443 1 if (0 != strcmp(submitfile, "\0"))
444 1 {
445 1     unlink(submitfile);
446 1 }
447 1 }
448 1 //delete the submitElementTree which is the iterator
449 1 delete submitElementTree;
450 1 // free the callloc memory
451 1 free(submitfile);
452 1 return 0;
453 1 }
454 1
455 1 /*****
456 1 **
456 1 ** Routine: getSubmitID
457 1 **
457 1 ** Inputs: None
458 1 **
458 1 ** Outputs: None
459 1 **
459 1 ** Return Codes:
460 1 **
460 1 ** Int - the submit ID
461 1 **
461 1 ** Purpose: Returns the submit ID.
462 1
463 1 *****/
464 1
465 1 //
466 1 int
467 1 EDRESsubmitObj::getSubmitID()
468 1 {
469 1     return submitID;
470 1 }
471 1
472 1 /*****
473 1 **
474 1 **
475 1 **
476 1 **
477 1 **
478 1 **
479 1 **
480 1 **
481 1 **
482 1 **
483 1 **
484 1 **
485 1 **
486 1 **
487 1 **
488 1 **
489 1 **
490 1 **
491 1 **
492 1 **
493 1 **
494 1 **
495 1 **
496 1 **
497 1 **
498 1 **
499 1 **
500 1 **
501 1 **
502 1 **
503 1 **
504 1 **
505 1 **
506 1 **
507 1 **
508 1 **
509 1 **
510 1 **
511 1 **
512 1 **
513 1 **
514 1 **
515 1 **
516 1 **
517 1 **
518 1 **
519 1 **
520 1 **
521 1 **
522 1 **
523 1 **
524 1 **
525 1 **
526 1 **
527 1 **
528 1 **
529 1 **
530 1 **
531 1 **
532 1 **
533 1 **
534 1 **
535 1 **
536 1 **
537 1 **
538 1 **
539 1 **
540 1 **
541 1 **
542 1 **
543 1 **
544 1 **
545 1 **
546 1 **
547 1 **
548 1 **
549 1 **
550 1 **
551 1 **
552 1 **
553 1 **
554 1 **
555 1 **
556 1 **
557 1 **
558 1 **
559 1 **
560 1 **
561 1 **
562 1 **
563 1 **
564 1 **
565 1 **
566 1 **
567 1 **
568 1 **
569 1 **
570 1 **
571 1 **
572 1 **
573 1 **
574 1 **
575 1 **
576 1 **
577 1 **
578 1 **
579 1 **
580 1 **
581 1 **
582 1 **
583 1 **
584 1 **
585 1 **
586 1 **
587 1 **
588 1 **
589 1 **
590 1 **
591 1 **
592 1 **
593 1 **
594 1 **
595 1 **
596 1 **
597 1 **
598 1 **
599 1 **
600 1 **
601 1 **
602 1 **
603 1 **
604 1 **
605 1 **
606 1 **
607 1 **
608 1 **
609 1 **
610 1 **
611 1 **
612 1 **
613 1 **
614 1 **
615 1 **
616 1 **
617 1 **
618 1 **
619 1 **
620 1 **
621 1 **
622 1 **
623 1 **
624 1 **
625 1 **
626 1 **
627 1 **
628 1 **
629 1 **
630 1 **
631 1 **
632 1 **
633 1 **
634 1 **
635 1 **
636 1 **
637 1 **
638 1 **
639 1 **
640 1 **
641 1 **
642 1 **
643 1 **
644 1 **
645 1 **
646 1 **
647 1 **
648 1 **
649 1 **
650 1 **
651 1 **
652 1 **
653 1 **
654 1 **
655 1 **
656 1 **
657 1 **
658 1 **
659 1 **
660 1 **
661 1 **
662 1 **
663 1 **
664 1 **
665 1 **
666 1 **
667 1 **
668 1 **
669 1 **
670 1 **
671 1 **
672 1 **
673 1 **
674 1 **
675 1 **
676 1 **
677 1 **
678 1 **
679 1 **
680 1 **
681 1 **
682 1 **
683 1 **
684 1 **
685 1 **
686 1 **
687 1 **
688 1 **
689 1 **
690 1 **
691 1 **
692 1 **
693 1 **
694 1 **
695 1 **
696 1 **
697 1 **
698 1 **
699 1 **
700 1 **
701 1 **
702 1 **
703 1 **
704 1 **
705 1 **
706 1 **
707 1 **
708 1 **
709 1 **
710 1 **
711 1 **
712 1 **
713 1 **
714 1 **
715 1 **
716 1 **
717 1 **
718 1 **
719 1 **
720 1 **
721 1 **
722 1 **
723 1 **
724 1 **
725 1 **
726 1 **
727 1 **
728 1 **
729 1 **
730 1 **
731 1 **
732 1 **
733 1 **
734 1 **
735 1 **
736 1 **
737 1 **
738 1 **
739 1 **
740 1 **
741 1 **
742 1 **
743 1 **
744 1 **
745 1 **
746 1 **
747 1 **
748 1 **
749 1 **
750 1 **
751 1 **
752 1 **
753 1 **
754 1 **
755 1 **
756 1 **
757 1 **
758 1 **
759 1 **
760 1 **
761 1 **
762 1 **
763 1 **
764 1 **
765 1 **
766 1 **
767 1 **
768 1 **
769 1 **
770 1 **
771 1 **
772 1 **
773 1 **
774 1 **
775 1 **
776 1 **
777 1 **
778 1 **
779 1 **
780 1 **
781 1 **
782 1 **
783 1 **
784 1 **
785 1 **
786 1 **
787 1 **
788 1 **
789 1 **
790 1 **
791 1 **
792 1 **
793 1 **
794 1 **
795 1 **
796 1 **
797 1 **
798 1 **
799 1 **
800 1 **
801 1 **
802 1 **
803 1 **
804 1 **
805 1 **
806 1 **
807 1 **
808 1 **
809 1 **
810 1 **
811 1 **
812 1 **
813 1 **
814 1 **
815 1 **
816 1 **
817 1 **
818 1 **
819 1 **
820 1 **
821 1 **
822 1 **
823 1 **
824 1 **
825 1 **
826 1 **
827 1 **
828 1 **
829 1 **
830 1 **
831 1 **
832 1 **
833 1 **
834 1 **
835 1 **
836 1 **
837 1 **
838 1 **
839 1 **
840 1 **
841 1 **
842 1 **
843 1 **
844 1 **
845 1 **
846 1 **
847 1 **
848 1 **
849 1 **
850 1 **
851 1 **
852 1 **
853 1 **
854 1 **
855 1 **
856 1 **
857 1 **
858 1 **
859 1 **
860 1 **
861 1 **
862 1 **
863 1 **
864 1 **
865 1 **
866 1 **
867 1 **
868 1 **
869 1 **
870 1 **
871 1 **
872 1 **
873 1 **
874 1 **
875 1 **
876 1 **
877 1 **
878 1 **
879 1 **
880 1 **
881 1 **
882 1 **
883 1 **
884 1 **
885 1 **
886 1 **
887 1 **
888 1 **
889 1 **
890 1 **
891 1 **
892 1 **
893 1 **
894 1 **
895 1 **
896 1 **
897 1 **
898 1 **
899 1 **
900 1 **
901 1 **
902 1 **
903 1 **
904 1 **
905 1 **
906 1 **
907 1 **
908 1 **
909 1 **
910 1 **
911 1 **
912 1 **
913 1 **
914 1 **
915 1 **
916 1 **
917 1 **
918 1 **
919 1 **
920 1 **
921 1 **
922 1 **
923 1 **
924 1 **
925 1 **
926 1 **
927 1 **
928 1 **
929 1 **
930 1 **
931 1 **
932 1 **
933 1 **
934 1 **
935 1 **
936 1 **
937 1 **
938 1 **
939 1 **
940 1 **
941 1 **
942 1 **
943 1 **
944 1 **
945 1 **
946 1 **
947 1 **
948 1 **
949 1 **
950 1 **
951 1 **
952 1 **
953 1 **
954 1 **
955 1 **
956 1 **
957 1 **
958 1 **
959 1 **
960 1 **
961 1 **
962 1 **
963 1 **
964 1 **
965 1 **
966 1 **
967 1 **
968 1 **
969 1 **
970 1 **
971 1 **
972 1 **
973 1 **
974 1 **
975 1 **
976 1 **
977 1 **
978 1 **
979 1 **
980 1 **
981 1 **
982 1 **
983 1 **
984 1 **
985 1 **
986 1 **
987 1 **
988 1 **
989 1 **
990 1 **
991 1 **
992 1 **
993 1 **
994 1 **
995 1 **
996 1 **
997 1 **
998 1 **
999 1 **
1000 1 **

```

```

499  ** Routine: setSubmtID
500
501  ** Inputs:  int submtcid - ID of the submtc job
502
503  ** Outputs: None
504
505  ** Return Codes:
506      None
507
508  ** Purpose: Sets the submtc ID using the specified parameter.
509
510
511  */
512
513  void
514  EPMRESsubmtID::setSubmtID(int submtcid)
515  {
516      submtCID = submtcid;
517  }
518
519  /*****
520  **
521  ** Routine: getSubmtCType
522
523  ** Inputs:  None
524
525  ** Outputs: None
526
527  ** Return Codes:
528      unsigned int - the submtc type
529
530  ** Purpose: Returns the submtc type.
531
532
533  */
534
535  unsigned int
536  EPMRESsubmtID::getSubmtCType()
537  {
538      return so_submtC_type;
539  }
540
541  /*****
542  **
543  ** Routine: setSubmtCType
544
545  ** Inputs:  unsigned int submtCtype - type of the submtc job
546
547  ** Outputs: None
548
549  ** Return Codes:
550      None
551
552  ** Purpose: Sets the submtc type using the specified parameter.
553
554
555  */
556
557  void
558  EPMRESsubmtID::setSubmtCType(unsigned int submtCtype)
559  {
560
561  }
562
563  /*****
564  **
565  ** Routine: getSubmtCType
566
567  ** Inputs:  None
568
569  ** Outputs: None
570
571  ** Return Codes:
572      None
573
574  ** Purpose: Returns the submtc type.
575
576
577  */
578
579  unsigned int
580  EPMRESsubmtID::getSubmtCType()
581  {
582      return so_submtC_type;
583  }
584
585  /*****
586  **
587  ** Routine: setSubmtCType
588
589  ** Inputs:  unsigned int submtCtype - type of the submtc job
590
591  ** Outputs: None
592
593  ** Return Codes:
594      None
595
596  ** Purpose: Sets the submtc type using the specified parameter.
597
598
599  */
600
601  void
602  EPMRESsubmtID::setSubmtCType(unsigned int submtCtype)
603  {
604
605  }
606
607  /*****
608  **
609  ** Routine: getSubmtCType
610
611  ** Inputs:  None
612
613  ** Outputs: None
614
615  ** Return Codes:
616      None
617
618  ** Purpose: Returns the submtc type.
619
620
621  */
622
623  unsigned int
624  EPMRESsubmtID::getSubmtCType()
625  {
626      return so_submtC_type;
627  }
628
629  /*****
630  **
631  ** Routine: setSubmtCType
632
633  ** Inputs:  unsigned int submtCtype - type of the submtc job
634
635  ** Outputs: None
636
637  ** Return Codes:
638      None
639
640  ** Purpose: Sets the submtc type using the specified parameter.
641
642
643  */
644
645  void
646  EPMRESsubmtID::setSubmtCType(unsigned int submtCtype)
647  {
648
649  }
650
651  /*****
652  **
653  ** Routine: getSubmtCType
654
655  ** Inputs:  None
656
657  ** Outputs: None
658
659  ** Return Codes:
660      None
661
662  ** Purpose: Returns the submtc type.
663
664
665  */
666
667  unsigned int
668  EPMRESsubmtID::getSubmtCType()
669  {
670      return so_submtC_type;
671  }
672
673  /*****
674  **
675  ** Routine: setSubmtCType
676
677  ** Inputs:  unsigned int submtCtype - type of the submtc job
678
679  ** Outputs: None
680
681  ** Return Codes:
682      None
683
684  ** Purpose: Sets the submtc type using the specified parameter.
685
686
687  */
688
689  void
690  EPMRESsubmtID::setSubmtCType(unsigned int submtCtype)
691  {
692
693  }
694
695  /*****
696  **
697  ** Routine: getSubmtCType
698
699  ** Inputs:  None
700
701  ** Outputs: None
702
703  ** Return Codes:
704      None
705
706  ** Purpose: Returns the submtc type.
707
708
709  */
710
711  unsigned int
712  EPMRESsubmtID::getSubmtCType()
713  {
714      return so_submtC_type;
715  }
716
717  /*****
718  **
719  ** Routine: setSubmtCType
720
721  ** Inputs:  unsigned int submtCtype - type of the submtc job
722
723  ** Outputs: None
724
725  ** Return Codes:
726      None
727
728  ** Purpose: Sets the submtc type using the specified parameter.
729
730
731  */
732
733  void
734  EPMRESsubmtID::setSubmtCType(unsigned int submtCtype)
735  {
736
737  }
738
739  /*****
740  **
741  ** Routine: getSubmtCType
742
743  ** Inputs:  None
744
745  ** Outputs: None
746
747  ** Return Codes:
748      None
749
750  ** Purpose: Returns the submtc type.
751
752
753  */
754
755  unsigned int
756  EPMRESsubmtID::getSubmtCType()
757  {
758      return so_submtC_type;
759  }
760
761  /*****
762  **
763  ** Routine: setSubmtCType
764
765  ** Inputs:  unsigned int submtCtype - type of the submtc job
766
767  ** Outputs: None
768
769  ** Return Codes:
770      None
771
772  ** Purpose: Sets the submtc type using the specified parameter.
773
774
775  */
776
777  void
778  EPMRESsubmtID::setSubmtCType(unsigned int submtCtype)
779  {
780
781  }
782
783  /*****
784  **
785  ** Routine: getSubmtCType
786
787  ** Inputs:  None
788
789  ** Outputs: None
790
791  ** Return Codes:
792      None
793
794  ** Purpose: Returns the submtc type.
795
796
797  */
798
799  unsigned int
800  EPMRESsubmtID::getSubmtCType()
801  {
802      return so_submtC_type;
803  }
804
805  /*****
806  **
807  ** Routine: setSubmtCType
808
809  ** Inputs:  unsigned int submtCtype - type of the submtc job
810
811  ** Outputs: None
812
813  ** Return Codes:
814      None
815
816  ** Purpose: Sets the submtc type using the specified parameter.
817
818
819  */
820
821  void
822  EPMRESsubmtID::setSubmtCType(unsigned int submtCtype)
823  {
824
825  }
826
827  /*****
828  **
829  ** Routine: getSubmtCType
830
831  ** Inputs:  None
832
833  ** Outputs: None
834
835  ** Return Codes:
836      None
837
838  ** Purpose: Returns the submtc type.
839
840
841  */
842
843  unsigned int
844  EPMRESsubmtID::getSubmtCType()
845  {
846      return so_submtC_type;
847  }
848
849  /*****
850  **
851  ** Routine: setSubmtCType
852
853  ** Inputs:  unsigned int submtCtype - type of the submtc job
854
855  ** Outputs: None
856
857  ** Return Codes:
858      None
859
860  ** Purpose: Sets the submtc type using the specified parameter.
861
862
863  */
864
865  void
866  EPMRESsubmtID::setSubmtCType(unsigned int submtCtype)
867  {
868
869  }
870
871  /*****
872  **
873  ** Routine: getSubmtCType
874
875  ** Inputs:  None
876
877  ** Outputs: None
878
879  ** Return Codes:
880      None
881
882  ** Purpose: Returns the submtc type.
883
884
885  */
886
887  unsigned int
888  EPMRESsubmtID::getSubmtCType()
889  {
890      return so_submtC_type;
891  }
892
893  /*****
894  **
895  ** Routine: setSubmtCType
896
897  ** Inputs:  unsigned int submtCtype - type of the submtc job
898
899  ** Outputs: None
900
901  ** Return Codes:
902      None
903
904  ** Purpose: Sets the submtc type using the specified parameter.
905
906
907  */
908
909  void
910  EPMRESsubmtID::setSubmtCType(unsigned int submtCtype)
911  {
912
913  }
914
915  /*****
916  **
917  ** Routine: getSubmtCType
918
919  ** Inputs:  None
920
921  ** Outputs: None
922
923  ** Return Codes:
924      None
925
926  ** Purpose: Returns the submtc type.
927
928
929  */
930
931  unsigned int
932  EPMRESsubmtID::getSubmtCType()
933  {
934      return so_submtC_type;
935  }
936
937  /*****
938  **
939  ** Routine: setSubmtCType
940
941  ** Inputs:  unsigned int submtCtype - type of the submtc job
942
943  ** Outputs: None
944
945  ** Return Codes:
946      None
947
948  ** Purpose: Sets the submtc type using the specified parameter.
949
950
951  */
952
953  void
954  EPMRESsubmtID::setSubmtCType(unsigned int submtCtype)
955  {
956
957  }
958
959  /*****
960  **
961  ** Routine: getSubmtCType
962
963  ** Inputs:  None
964
965  ** Outputs: None
966
967  ** Return Codes:
968      None
969
970  ** Purpose: Returns the submtc type.
971
972
973  */
974
975  unsigned int
976  EPMRESsubmtID::getSubmtCType()
977  {
978      return so_submtC_type;
979  }
980
981  /*****
982  **
983  ** Routine: setSubmtCType
984
985  ** Inputs:  unsigned int submtCtype - type of the submtc job
986
987  ** Outputs: None
988
989  ** Return Codes:
990      None
991
992  ** Purpose: Sets the submtc type using the specified parameter.
993
994
995  */
996
997  void
998  EPMRESsubmtID::setSubmtCType(unsigned int submtCtype)
999  {
1000
1001  }
1002
1003  /*****
1004  **
1005  ** Routine: getSubmtCType
1006
1007  ** Inputs:  None
1008
1009  ** Outputs: None
1010
1011  ** Return Codes:
1012      None
1013
1014  ** Purpose: Returns the submtc type.
1015
1016
1017  */
1018
1019  unsigned int
1020  EPMRESsubmtID::getSubmtCType()
1021  {
1022      return so_submtC_type;
1023  }
1024
1025  /*****
1026  **

```

```

560 1      so_submitt_type = submittType;
561
562 }
563
564 //*****
565 **
566 ** Routine: getExecutionFlags
567 **
568 ** Inputs:  None
569 **
570 ** Outputs: None
571 **
572 ** Return Codes:
573 **      unsigned int - the execution flags
574 **
575 ** Purpose: Returns the execution flags.
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000

```



```

620 .....
621 */
622 .....
623 boolean ly
624 EDRESsubmitObj::getNoVmCheck()
625 {
626     return so_no_vm_check;
627 }
628 .....
629 .....
630 .....
631 ** Routine: setNoVmCheck
632 .....
633 ** Inputs: boolean ly noVmCheck - no vm check boolean
634 .....
635 ** Outputs: None
636 .....
637 ** Return Codes:
638     None
639 .....
640 ** Purpose: Sets the No-Vm-Check boolean to the specified parameter.
641 .....
642 .....
643 */
644 void
645 EDRESsubmitObj::setNoVmCheck(boolean ly noVmCheck)
646 {
647     so_no_vm_check = noVmCheck;
648 }
649 .....
650 .....
651 .....
652 .....
653 ** Routine: getPrePhaseExecutable
654 .....
655 ** Inputs: int maxSize - max size to copy to buffer
656 .....
657 ** Outputs: char *executable - executable to run in the pre-phase
658 .....
659 ** Return Codes:
660     None
661 .....
662 ** Purpose: Returns the executable to run in the pre-phase in the
663     specified buffer.
664 .....
665 .....
666 */
667 .....
668 void
669 EDRESsubmitObj::getPrePhaseExecutable(char *executable, int maxSize)
670 {
671     if (executable != NULL && pre_phase_executable != NULL)
672     {
673         strncpy(executable, pre_phase_executable, maxSize);
674         executable[maxSize - 1] = 0;
675     }
676     else if (executable != NULL)
677     {
678         executable[0] = 0;
679     }
680 }
681 .....

```

```

682 .....
683 .....
684 ** Routine: setPrePhaseExecutable
685 .....
686 ** Inputs: char *executable - executable to run in the pre-phase
687 .....
688 ** Outputs: None
689 .....
690 ** Return Codes:
691     None
692 .....
693 ** Purpose: Sets the executable to run during the pre-phase using
694     the specified parameter.
695 .....
696 .....
697 */
698 void
699 EDRESsubmitObj::setPrePhaseExecutable(char *executable)
700 {
701     if (executable != NULL)
702     {
703         if (pre_phase_executable != NULL)
704             free(pre_phase_executable);
705         pre_phase_executable = strdup(executable);
706     }
707 }
708 .....
709 .....
710 .....
711 .....
712 .....
713 ** Routine: getPrePhaseArgs
714 .....
715 ** Inputs: None
716 .....
717 ** Outputs: char **args - args to use in the pre-phase
718 .....
719 ** Return Codes:
720     None
721 .....
722 ** Purpose: Returns the args to use in the pre-phase in the
723     specified buffer.
724 .....
725 .....
726 */
727 .....
728 void
729 EDRESsubmitObj::getPrePhaseArgs(char **args)
730 {
731     // Should make real copies of args and env
732     if (args != NULL)
733     {
734         *args = prePhase_args;
735     }
736 }
737 .....
738 .....
739 ** Routine: setPrePhaseArgs
740 .....
741 ** Inputs: char *args - args to use in the pre-phase
742 .....

```

742	** Outputs:	None
743	** Return Codes:	None
744	**	
745	**	
746	**	
747	** Purpose:	Sets the args to use during the pre-phase using
748	**	the specified parameter.
749	**	
750	**	
751	*/	
752	void	
753	EDMResubmitObj::setPrePhaseArgs(char **args)	
754	{	
755	// Should make real copies of args and env	
756	if (args == NULL)	
757	pre_phase_argv = args;	
758	}	
759	}	
760	/*	
761	*****	
762	** Routine:	getPrePhaseArgv
763	**	
764	** Inputs:	None
765	**	
766	** Outputs:	char **env - env to use in the pre-phase
767	**	
768	** Return Codes:	None
769	**	
770	**	
771	** Purpose:	Returns the env to use in the pre-phase in the
772	**	specified buffer.
773	**	
774	*****	
775	*/	
776	void	
777	EDMResubmitObj::getPrePhaseArgv(char **env)	
778	{	
779	// Should make real copies of args and env	
780	if (env == NULL)	
781	*env = pre_phase_argv;	
782	}	
783	}	
784	/*	
785	*****	
786	*/	
787	*****	
788	** Routine:	setPrePhaseArgv
789	**	
790	** Inputs:	char **env - env to use in the pre-phase
791	**	
792	** Outputs:	None
793	**	
794	** Return Codes:	None
795	**	
796	**	
797	** Purpose:	Sets the env to use during the pre-phase using
798	**	the specified parameter.
799	**	
800	*****	
801	*/	

802	void	
803	EDMResubmitObj::setPrePhaseArgv(char **env)	
804	{	
805	// Should make real copies of args and env	
806	if (env == NULL)	
807	pre_phase_argv = env;	
808	}	
809	}	
810	/*	
811	*****	
812	** Routine:	getExecutePhaseExecutable
813	**	
814	** Inputs:	int maxsize - maximum size to copy into the executable
815	**	buffer
816	**	
817	** Outputs:	char *executable - executable to run in the execute phase
818	**	
819	** Return Codes:	None
820	**	
821	** Purpose:	Returns the executable to run in the execute-phase in the
822	**	specified buffer.
823	**	
824	*****	
825	*/	
826	void	
827	EDMResubmitObj::getExecutePhaseExecutable(char *executable, int maxsize)	
828	{	
829	if (executable == NULL && execute_override_phase_executable ==	
830	NULL)	
831	strcpy(
832	executable, execute_override_phase_executable, maxsize);	
833	executable[maxsize - 1] = 0;	
834	}	
835	else if (executable != NULL)	
836	executable[0] = 0;	
837	}	
838	}	
839	}	
840	}	
841	/*	
842	*****	
843	** Routine:	setExecutePhaseExecutable
844	**	
845	** Inputs:	char *executable - executable to run in the execute-phase
846	**	
847	** Outputs:	None
848	**	
849	** Return Codes:	None
850	**	
851	**	
852	** Purpose:	Sets the executable to run during the execute-phase using
853	**	the specified parameter.
854	**	
855	*****	
856	*/	
857	void	
858	EDMResubmitObj::setExecutePhaseExecutable(char *executable)	
859	{	

```

861 1 {
862 1     if (executable != NULL)
863 1     {
864 2         if (execute_override_phase_executable != NULL)
865 2             free(execute_override_phase_executable);
866 2         execute_override_phase_executable = strdup(executable);
867 2     }
868 1 }
869 1
870 1 /*****
871 1
872 1 **
873 1 ** Routine: getExecutePhaseArgs
874 1 **
875 1 ** Inputs: None
876 1 **
877 1 ** Outputs: char **args - args to use in the execute-phase
878 1 **
879 1 ** Return Codes:
880 1 **     None
881 1 **
882 1 ** Purpose: Returns the args to use in the execute-phase in the
883 1 **           specified buffer.
884 1 **
885 1 *****/
886 1 */
887 1 void
888 1 ERMEShmbtObj::getExecutePhaseArgs(char **args)
889 1 {
890 1     // Should make real copies of args and env
891 1     if (args != NULL)
892 1         *args = execute_override_phase_argv;
893 1 }
894 1
895 1 /*****
896 1
897 1 **
898 1 ** Routine: setExecutePhaseArgs
899 1 **
900 1 ** Inputs: char **args - args to use in the execute-phase
901 1 **
902 1 ** Outputs: None
903 1 **
904 1 ** Return Codes:
905 1 **     None
906 1 **
907 1 ** Purpose: Sets the args to use during the execute-phase using
908 1 **           the specified parameter.
909 1 **
910 1 *****/
911 1 */
912 1 void
913 1 ERMEShmbtObj::setExecutePhaseArgs(char **args)
914 1 {
915 1     // Should make real copies of args and env
916 1     if (args != NULL)
917 1         execute_override_phase_argv = args;
918 1 }
919 1
920 1 /*****
921 1
922 1 *****/

```

```

922 1 **
923 1 ** Routine: getExecutePhaseEnv
924 1 **
925 1 ** Inputs: None
926 1 **
927 1 ** Outputs: char **env - env to use in the execute-phase
928 1 **
929 1 ** Return Codes:
930 1 **     None
931 1 **
932 1 ** Purpose: Returns the env to use in the execute-phase in the
933 1 **           specified buffer.
934 1 **
935 1 *****/
936 1 */
937 1 void
938 1 ERMEShmbtObj::getExecutePhaseEnv(char **env)
939 1 {
940 1     // Should make real copies of args and env
941 1     if (env != NULL)
942 1         *env = execute_override_phase_env;
943 1 }
944 1
945 1 /*****
946 1
947 1 **
948 1 ** Routine: setExecutePhaseEnv
949 1 **
950 1 ** Inputs: char **env - env to use in the execute-phase
951 1 **
952 1 ** Outputs: None
953 1 **
954 1 ** Return Codes:
955 1 **     None
956 1 **
957 1 ** Purpose: Sets the env to use during the execute-phase using
958 1 **           the specified parameter.
959 1 **
960 1 *****/
961 1 */
962 1 void
963 1 ERMEShmbtObj::setExecutePhaseEnv(char **env)
964 1 {
965 1     // Should make real copies of args and env
966 1     if (env != NULL)
967 1         execute_override_phase_env = env;
968 1 }
969 1
970 1 /*****
971 1
972 1 **
973 1 ** Routine: getPostPhaseExecutable
974 1 **
975 1 ** Inputs: int maxsize - maximum size to copy into the executable
976 1 **           buffer
977 1 **
978 1 ** Outputs: char *executable - executable to run in the post-phase
979 1 **
980 1 ** Return Codes:
981 1 **     None
982 1 **
983 1 ** Purpose: Returns the executable to run in the post-phase in the
984 1 **           specified buffer.
985 1 **
986 1 *****/

```

```

983 ** specified buffer.
984 *****
985
986 */
987
988 void
989 EDRES::submitObj::getPostPhaseExecutable(char *executable, int maxsize)
990 {
991     if (executable != NULL && post_phase_executable != NULL)
992     {
993         strncpy(executable, post_phase_executable, maxsize);
994         executable[maxsize - 1] = 0;
995     }
996     else if (executable != NULL)
997     {
998         executable[0] = 0;
999     }
1000 }
1001
1002 /*****
1003 ** Routine: setPostPhaseExecutable
1004 **
1005 ** Inputs: char *executable - executable to run in the post-phase
1006 ** Outputs: None
1007 ** Return Codes: None
1008 **
1009 ** Purpose: Sets the executable to run during the post-phase using
1010             the specified parameter.
1011 *****/
1012 void
1013 EDRES::submitObj::setPostPhaseExecutable(char *executable)
1014 {
1015     if (executable != NULL)
1016     {
1017         if (post_phase_executable != NULL)
1018             Free(post_phase_executable);
1019         post_phase_executable = strdup(executable);
1020     }
1021 }
1022
1023 void
1024 EDRES::submitObj::setPostPhaseEnv(char **args)
1025 {
1026     if (args != NULL)
1027     {
1028         // Should make real copies of args and env
1029         if (args != NULL)
1030             post_phase_argv = args;
1031     }
1032 }
1033
1034 /*****
1035 ** Routine: getPostPhaseEnv
1036 **
1037 ** Inputs: None
1038 ** Outputs: char **env - env to use in the post-phase
1039 ** Return Codes: None
1040 **
1041 ** Purpose: Returns the env to use in the post-phase in the
1042             specified buffer.
1043 *****/
1044 void
1045 EDRES::submitObj::getPostPhaseEnv(char **env)
1046 {
1047     if (env != NULL)
1048         *env = post_phase_env;
1049 }
1050
1051 /*****
1052 ** Routine: setPostPhaseArgs
1053 **
1054 ** Inputs: char **args - args to use in the post-phase
1055 ** Outputs: None
1056 ** Return Codes: None
1057 **
1058 ** Purpose: Sets the args to use during the post-phase using
1059             the specified parameter.
1060 *****/
1061 void
1062 EDRES::submitObj::setPostPhaseArgs(char **args)
1063 {
1064     if (args != NULL)
1065         post_phase_argv = args;
1066 }
1067
1068 /*****
1069 ** Routine: getPostPhaseArgs
1070 **
1071 ** Inputs: None
1072 ** Outputs: char **args - args to use in the post-phase
1073 ** Return Codes: None
1074 **
1075 ** Purpose: Returns the args to use in the post-phase in the
1076             specified buffer.
1077 *****/
1078 void
1079 EDRES::submitObj::getPostPhaseArgs(char **args)
1080 {
1081     if (args != NULL)
1082         *args = post_phase_argv;
1083 }
1084
1085 /*****
1086 ** Routine: getPostPhaseEnv
1087 **
1088 ** Inputs: None
1089 ** Outputs: char **env - env to use in the post-phase
1090 ** Return Codes: None
1091 **
1092 ** Purpose: Returns the env to use in the post-phase in the
1093             specified buffer.
1094 *****/
1095 void
1096 EDRES::submitObj::getPostPhaseEnv(char **env)
1097 {
1098     if (env != NULL)
1099         *env = post_phase_env;
1100 }
1101
1102 /*****
1103 ** Routine: setPostPhaseArgs
1104 **
1105 ** Inputs: char **args - args to use in the post-phase
1106 ** Outputs: None
1107 ** Return Codes: None
1108 **
1109 ** Purpose: Sets the args to use during the post-phase using
1110             the specified parameter.
1111 *****/
1112 void
1113 EDRES::submitObj::setPostPhaseArgs(char **args)
1114 {
1115     if (args != NULL)
1116         post_phase_argv = args;
1117 }
1118
1119 /*****
1120 ** Routine: getPostPhaseArgs
1121 **
1122 ** Inputs: None
1123 ** Outputs: char **args - args to use in the post-phase
1124 ** Return Codes: None
1125 **
1126 ** Purpose: Returns the args to use in the post-phase in the
1127             specified buffer.
1128 *****/
1129 void
1130 EDRES::submitObj::getPostPhaseArgs(char **args)
1131 {
1132     if (args != NULL)
1133         *args = post_phase_argv;
1134 }

```

```

1045
1046 */
1047
1048 void
1049 EDRES::submitObj::getPostPhaseArgs(char **args)
1050 {
1051     if (args != NULL)
1052     {
1053         // Should make real copies of args and env
1054         if (args != NULL)
1055             post_phase_argv = args;
1056     }
1057 }
1058
1059 /*****
1060 ** Routine: setPostPhaseArgs
1061 **
1062 ** Inputs: char **args - args to use in the post-phase
1063 ** Outputs: None
1064 ** Return Codes: None
1065 **
1066 ** Purpose: Sets the args to use during the post-phase using
1067             the specified parameter.
1068 *****/
1069 void
1070 EDRES::submitObj::setPostPhaseArgs(char **args)
1071 {
1072     if (args != NULL)
1073         post_phase_argv = args;
1074 }
1075
1076 /*****
1077 ** Routine: getPostPhaseArgs
1078 **
1079 ** Inputs: None
1080 ** Outputs: char **args - args to use in the post-phase
1081 ** Return Codes: None
1082 **
1083 ** Purpose: Returns the args to use in the post-phase in the
1084             specified buffer.
1085 *****/
1086 void
1087 EDRES::submitObj::getPostPhaseArgs(char **args)
1088 {
1089     if (args != NULL)
1090         *args = post_phase_argv;
1091 }
1092
1093 /*****
1094 ** Routine: getPostPhaseEnv
1095 **
1096 ** Inputs: None
1097 ** Outputs: char **env - env to use in the post-phase
1098 ** Return Codes: None
1099 **
1100 ** Purpose: Returns the env to use in the post-phase in the
1101             specified buffer.
1102 *****/
1103 void
1104 EDRES::submitObj::getPostPhaseEnv(char **env)
1105 {
1106     if (env != NULL)
1107         *env = post_phase_env;
1108 }
1109
1110 /*****
1111 ** Routine: setPostPhaseArgs
1112 **
1113 ** Inputs: char **args - args to use in the post-phase
1114 ** Outputs: None
1115 ** Return Codes: None
1116 **
1117 ** Purpose: Sets the args to use during the post-phase using
1118             the specified parameter.
1119 *****/
1120 void
1121 EDRES::submitObj::setPostPhaseArgs(char **args)
1122 {
1123     if (args != NULL)
1124         post_phase_argv = args;
1125 }
1126
1127 /*****
1128 ** Routine: getPostPhaseArgs
1129 **
1130 ** Inputs: None
1131 ** Outputs: char **args - args to use in the post-phase
1132 ** Return Codes: None
1133 **
1134 ** Purpose: Returns the args to use in the post-phase in the
1135             specified buffer.
1136 *****/
1137 void
1138 EDRES::submitObj::getPostPhaseArgs(char **args)
1139 {
1140     if (args != NULL)
1141         *args = post_phase_argv;
1142 }

```

```

1106 /*****
1107 **
1108 ** Routine: setPostPhaseEnv
1109 **
1110 ** Inputs:  char **env - env to use in the post-phase
1111 **
1112 ** Outputs: None
1113 **
1114 ** Return Codes:
1115 **      None
1116 **
1117 ** Purpose: Sets the env to use during the post-phase using
1118 **           the specified parameter.
1119 **
1120 *****/
1121 **/
1122 void
1123 EPMESubmitObj::setPostPhaseEnv(char **env)
1124 {
1125     // Should make real copies of args and env
1126     if (env != NULL)
1127     {
1128         post_phase_env = env;
1129     }
1130 /*****/
1131 **/
1132 ** Routine: isBackupAdmin
1133 **
1134 ** Inputs:  None
1135 **
1136 ** Outputs: None
1137 **
1138 ** Return Codes:
1139 **      boolean, cv - TRUE if the user is backup admin
1140 **
1141 ** Purpose: Returns the backup admin boolean.
1142 **
1143 *****/
1144 **/
1145 boolean, cv
1146 EPMESubmitObj::isBackupAdmin()
1147 {
1148     return so_backup_administrator;
1149 }
1150 /*****/
1151 **/
1152 ** Routine: setIsBackupAdmin
1153 **
1154 ** Inputs:  boolean, cv isAdmin - Is backup admin
1155 **
1156 ** Outputs: None
1157 **
1158 ** Return Codes:
1159 **      None
1160 **
1161 ** Purpose: Sets the backup admin boolean to the specified parameter.
1162 **
1163 *****/
1164 **/

```

Page 240 of 248	File: restore/EDMRESumInfo/cc 20	File: Jan 04 16:36:25 2008
1267	*/	
1169	void	
1170	EDMRESumInfo::setIsBackupAdmin(BOOL bAdmin)	
1171	{	
1172	so_BackupAdminiator = IsAdmin;	
1173	}	
1174	
1175	
1176	
1177	Routine: IsSourceSystemAdmin	
1178	Inputs: None	
1179	Outputs: None	
1180	Return Codes:	
1181	boolean_Ty - TRUE if the user is admin on client system	
1182	Purpose: Returns the backup admin boolean.	
1183	
1184	
1185	
1186	
1187	
1188	
1189	*/	
1191	boolean_Ty	
1192	EDMRESumInfo::IsSourceSystemAdmin()	
1193	{	
1194	return so_src_sys_admin;	
1195	}	
1197	
1198	
1199	
1200	
1201	
1202	Routine: getIsBackupAdmin	
1203	Inputs: boolean_Ty IsAdmin - is admin on client machine	
1204	Outputs: None	
1205	Return Codes:	
1206	None	
1207	Purpose: Sets the admin-on-client boolean to the specified	
1208	parameter.	
1209	
1210	
1211	
1212	*/	
1213	void	
1214	EDMRESumInfo::setIsSourceSystemAdmin(BOOL bAdmin)	
1215	{	
1216	so_src_sys_admin = IsAdmin;	
1217	}	
1218	
1219	
1220	
1221	Routine: IsAdminOnAdmin	
1222	Inputs: None	
1223	Outputs: None	
1224	
1225	

1226	**	Return Codes:	
1228	**	boolean_ly - TRUE if the user is destination machine	admin
1229	**	Purpose: Returns the destination admin boolean.	
1230	**		
1231	**		
1232	**		
1233	*/		
1235	boolean_ly	EDMESubmitObj::IsDestinationAdmin()	
1236	{		
1237	return so_dec_sys_admin;		
1238	}		
1239	/*		
1241	/*		
1242	**	Routine: setIsDestinationAdmin	
1243	**		
1244	**	Inputs: boolean_ly IsdecAdmin - is destination host admin	
1245	**		
1246	**	Outputs: None	
1247	**	Return Codes:	
1248	**	None	
1249	**		
1251	**	Purpose: Sets the destination host admin boolean to the specified parameter.	
1252	**		
1253	**		
1254	**		
1255	*/		
1257	void	EDMESubmitObj::setIsDestinationAdmin(boolean_ly IsdecAdmin)	
1258	{		
1259	so_dec_sys_admin = IsdecAdmin;		
1260	}		
1261	/*		
1263	/*		
1264	**	Routine: getUserId	
1265	**		
1266	**	Inputs: None	
1267	**	Outputs: None	
1268	**		
1269	**	Return Codes:	
1270	**	None	
1271	**		
1272	**	Purpose: Returns the user ID of the user doing the restore.	
1273	**	uid_t - the userid of the user	
1274	**		
1275	**		
1276	**		
1277	*/		
1279	uid_t	EDMESubmitObj::getUserId()	
1280	{		
1281	return so_human_uid;		
1282	}		
1283	/*		

1285	/*		
1286	**	Routine: setUserId	
1287	**		
1288	**	Inputs: uid_t uid - userid of user	
1289	**		
1290	**	Outputs: None	
1291	**	Return Codes:	
1292	**	None	
1293	**		
1294	**	Purpose: Sets the userid to the specified parameter.	
1295	**		
1296	**		
1297	**		
1298	**		
1299	*/		
1301	void	EDMESubmitObj::setUserId(uid_t uid)	
1302	{		
1303	so_human_uid = uid;		
1304	}		
1305	/*		
1307	/*		
1308	**	Routine: getEffectiveUserId	
1309	**		
1310	**	Inputs: None	
1311	**	Outputs: None	
1312	**	Return Codes:	
1313	**	None	
1314	**		
1315	**	Purpose: Returns the effective user ID of the user doing the restore.	
1316	**	uid_t - the effective userid of the user	
1317	**		
1318	**		
1319	**		
1320	**		
1321	*/		
1323	uid_t	EDMESubmitObj::getEffectiveUserId()	
1324	{		
1325	return so_effective_uid;		
1326	}		
1327	/*		
1329	/*		
1330	**	Routine: setEffectiveUserId	
1331	**		
1332	**	Inputs: uid_t uid - effective userid of user	
1333	**		
1334	**	Outputs: None	
1335	**	Return Codes:	
1336	**	None	
1337	**		
1338	**	Purpose: Sets the effective userid to the specified parameter.	
1339	**		
1340	**		
1341	**		
1342	**		
1343	*/		

```

1345 void
1346 EDRMSsubnlObj::setEffectiveUserID(uid_t uid)
1347 {
1348     so_effective_uid = uid;
1349 }
1350
1351 /*****
1352 **
1353 ** Routine: getUsername
1354 ** Inputs:  int maxsize - size of the buffer passed in
1355 ** Outputs: char *username - name of the user
1356 **
1357 ** Return Codes:
1358 **             None
1359 **
1360 ** Purpose: Returns the name of the user in the specified buffer.
1361 **
1362 *****/
1363 void
1364 EDRMSsubnlObj::getUsername(char *username, int maxsize)
1365 {
1366     if (username != NULL && so_human_uidname != NULL)
1367     {
1368         strncpy(username, so_human_uidname, maxsize);
1369         username[maxsize - 1] = 0;
1370     }
1371     else if (username != NULL)
1372     {
1373         username[0] = 0;
1374     }
1375 }
1376
1377 /*****
1378 **
1379 ** Routine: setUsername
1380 ** Inputs:  char *username - name of the user
1381 ** Outputs: None
1382 **
1383 ** Return Codes:
1384 **             None
1385 **
1386 ** Purpose: Sets the name of the user in the specified parameter.
1387 **
1388 *****/
1389 void
1390 EDRMSsubnlObj::setUsername(char *username)
1391 {
1392     if (username != NULL)
1393     {
1394         if (so_human_uidname != NULL)
1395             free(so_human_uidname);
1396         so_human_uidname = strdup(username);
1397     }
1398 }
1399
1400 /*****
1401 **
1402 ** Routine: getEffectiveUsername
1403 ** Inputs:  int maxsize - size of the buffer passed in
1404 ** Outputs: char *username - name of the user
1405 **
1406 ** Return Codes:
1407 **             None
1408 **
1409 ** Purpose: Returns the name of the user in the specified buffer.
1410 **
1411 *****/
1412 void
1413 EDRMSsubnlObj::getEffectiveUsername(char *username, int maxsize)
1414 {
1415     if (username != NULL && so_effective_uidname != NULL)
1416     {
1417         strncpy(username, so_effective_uidname, maxsize);
1418         username[maxsize - 1] = 0;
1419     }
1420     else if (username != NULL)
1421     {
1422         username[0] = 0;
1423     }
1424 }
1425
1426 /*****
1427 **
1428 ** Routine: setEffectiveUsername
1429 ** Inputs:  char *username - name of the user
1430 ** Outputs: None
1431 **
1432 ** Return Codes:
1433 **             None
1434 **
1435 ** Purpose: Sets the name of the user in the specified parameter.
1436 **
1437 *****/
1438 void
1439 EDRMSsubnlObj::setEffectiveUsername(char *username)
1440 {
1441     if (username != NULL)
1442     {
1443         if (so_effective_uidname != NULL)
1444             free(so_effective_uidname);
1445         so_effective_uidname = strdup(username);
1446     }
1447 }
1448
1449 /*****

```

```

1450     }
1451 }
1452
1453 /*****
1454 **
1455 ** Routine: getEffectiveUsername
1456 ** Inputs:  int maxsize - size of the buffer passed in
1457 ** Outputs: char *username - name of the user
1458 **
1459 ** Return Codes:
1460 **             None
1461 **
1462 ** Purpose: Returns the name of the user in the specified buffer.
1463 **
1464 *****/
1465 void
1466 EDRMSsubnlObj::getEffectiveUsername(char *username, int maxsize)
1467 {
1468     if (username != NULL && so_effective_uidname != NULL)
1469     {
1470         strncpy(username, so_effective_uidname, maxsize);
1471         username[maxsize - 1] = 0;
1472     }
1473     else if (username != NULL)
1474     {
1475         username[0] = 0;
1476     }
1477 }
1478
1479 /*****
1480 **
1481 ** Routine: setEffectiveUsername
1482 ** Inputs:  char *username - name of the user
1483 ** Outputs: None
1484 **
1485 ** Return Codes:
1486 **             None
1487 **
1488 ** Purpose: Sets the name of the user in the specified parameter.
1489 **
1490 *****/
1491 void
1492 EDRMSsubnlObj::setEffectiveUsername(char *username)
1493 {
1494     if (username != NULL)
1495     {
1496         if (so_effective_uidname != NULL)
1497             free(so_effective_uidname);
1498         so_effective_uidname = strdup(username);
1499     }
1500 }
1501
1502 /*****

```

```

1469 ** Routine: getMarkSummary
1470
1471 ** Inputs: None
1472
1473 ** Outputs: struct mark_summary *markptr - place to put mark summary
1474
1475 ** Return Codes: None
1476
1477 ** Purpose: Returns the mark summary in the specified buffer.
1478
1479
1480
1481 */
1482
1483 void
1484 EDPRSsubnlCOB::getMarkSummary(struct mark_summary *markptr)
1485 {
1486     if (markptr != NULL)
1487     {
1488         memcpy(markptr, kso_total_subnl_summary, sizeof(
1489             struct mark_summary));
1490     }
1491 }
1492
1493
1494
1495 ** Routine: setMarkSummary
1496
1497 ** Inputs: struct mark_summary *markptr - place to put the mark
1498             summary
1499
1500 ** Outputs: None
1501
1502 ** Return Codes: None
1503
1504 ** Purpose: Sets the mark summary using the specified buffer.
1505
1506
1507 */
1508 void
1509 EDPRSsubnlCOB::setMarkSummary(struct mark_summary *markptr)
1510 {
1511     if (markptr != NULL)
1512     {
1513         memcpy(kso_total_subnl_summary, markptr, sizeof(
1514             struct mark_summary));
1515     }
1516 }
1517
1518
1519
1520 ** Routine: getVolumeList
1521
1522 ** Inputs: None
1523
1524 ** Outputs: ebyl_voidlist_t *volutr - place to put the volume list
1525
1526 ** Return Codes: None
1527
1528 ** Purpose: Returns the volume list in the specified buffer.
1529
1530
1531
1532
1533
1534
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000

```

```

1527 */
1528
1529 void
1530 EDRESsubmitObj::getVolumeList(ebv_volidlist_ty **volptr)
1531 {
1532     if (volptr != NULL)
1533         *volptr = so_total_volume_list;
1534 }
1535
1536
1537
1538
1539
1540
1541
1542
1543
1544
1545
1546
1547
1548
1549
1550
1551
1552
1553
1554
1555
1556
1557
1558
1559
1560
1561
1562
1563
1564
1565
1566
1567
1568
1569
1570
1571
1572
1573
1574
1575
1576
1577
1578
1579
1580
1581
1582
1583
1584
1585
1586
1587
1588
1589
1590
1591
1592
1593
1594
1595
1596
1597
1598
1599
1600
1601
1602
1603
1604
1605
1606
1607
1608
1609
1610
1611
1612
1613
1614
1615
1616
1617
1618
1619
1620
1621
1622
1623
1624
1625
1626
1627
1628
1629
1630
1631
1632
1633
1634
1635
1636
1637
1638
1639
1640
1641
1642
1643
1644
1645
1646
1647
1648
1649
1650
1651
1652
1653
1654
1655
1656
1657
1658
1659
1660
1661
1662
1663
1664
1665
1666
1667
1668
1669
1670
1671
1672
1673
1674
1675
1676
1677
1678
1679
1680
1681
1682
1683
1684
1685
1686
1687
1688
1689
1690
1691
1692
1693
1694
1695
1696
1697
1698
1699
1700
1701
1702
1703
1704
1705
1706
1707
1708
1709
1710
1711
1712
1713
1714
1715
1716
1717
1718
1719
1720
1721
1722
1723
1724
1725
1726
1727
1728
1729
1730
1731
1732
1733
1734
1735
1736
1737
1738
1739
1740
1741
1742
1743
1744
1745
1746
1747
1748
1749
1750
1751
1752
1753
1754
1755
1756
1757
1758
1759
1760
1761
1762
1763
1764
1765
1766
1767
1768
1769
1770
1771
1772
1773
1774
1775
1776
1777
1778
1779
1780
1781
1782
1783
1784
1785
1786
1787
1788
1789
1790
1791
1792
1793
1794
1795
1796
1797
1798
1799
1800
1801
1802
1803
1804
1805
1806
1807
1808
1809
1810
1811
1812
1813
1814
1815
1816
1817
1818
1819
1820
1821
1822
1823
1824
1825
1826
1827
1828
1829
1830
1831
1832
1833
1834
1835
1836
1837
1838
1839
1840
1841
1842
1843
1844
1845
1846
1847
1848
1849
1850
1851
1852
1853
1854
1855
1856
1857
1858
1859
1860
1861
1862
1863
1864
1865
1866
1867
1868
1869
1870
1871
1872
1873
1874
1875
1876
1877
1878
1879
1880
1881
1882
1883
1884
1885
1886
1887
1888
1889
1890
1891
1892
1893
1894
1895
1896
1897
1898
1899
1900
1901
1902
1903
1904
1905
1906
1907
1908
1909
1910
1911
1912
1913
1914
1915
1916
1917
1918
1919
1920
1921
1922
1923
1924
1925
1926
1927
1928
1929
1930
1931
1932
1933
1934
1935
1936
1937
1938
1939
1940
1941
1942
1943
1944
1945
1946
1947
1948
1949
1950
1951
1952
1953
1954
1955
1956
1957
1958
1959
1960
1961
1962
1963
1964
1965
1966
1967
1968
1969
1970
1971
1972
1973
1974
1975
1976
1977
1978
1979
1980
1981
1982
1983
1984
1985
1986
1987
1988
1989
1990
1991
1992
1993
1994
1995
1996
1997
1998
1999
2000
2001
2002
2003
2004
2005
2006
2007
2008
2009
2010
2011
2012
2013
2014
2015
2016
2017
2018
2019
2020
2021
2022
2023
2024
2025
2026
2027
2028
2029
2030
2031
2032
2033
2034
2035
2036
2037
2038
2039
2040
2041
2042
2043
2044
2045
2046
2047
2048
2049
2050
2051
2052
2053
2054
2055
2056
2057
2058
2059
2060
2061
2062
2063
2064
2065
2066
2067
2068
2069
2070
2071
2072
2073
2074
2075
2076
2077
2078
2079
2080
2081
2082
2083
2084
2085
2086
2087
2088
2089
2090
2091
2092
2093
2094
2095
2096
2097
2098
2099
2100
2101
2102
2103
2104
2105
2106
2107
2108
2109
2110
2111
2112
2113
2114
2115
2116
2117
2118
2119
2120
2121
2122
2123
2124
2125
2126
2127
2128
2129
2130
2131
2132
2133
2134
2135
2136
2137
2138
2139
2140
2141
2142
2143
2144
2145
2146
2147
2148
2149
2150
2151
2152
2153
2154
2155
2156
2157
2158
2159
2160
2161
2162
2163
2164
2165
2166
2167
2168
2169
2170
2171
2172
2173
2174
2175
2176
2177
2178
2179
2180
2181
2182
2183
2184
2185
2186
2187
2188
2189
2190
2191
2192
2193
2194
2195
2196
2197
2198
2199
2200
2201
2202
2203
2204
2205
2206
2207
2208
2209
2210
2211
2212
2213
2214
2215
2216
2217
2218
2219
2220
2221
2222
2223
2224
2225
2226
2227
2228
2229
2230
2231
2232
2233
2234
2235
2236
2237
2238
2239
2240
2241
2242
2243
2244
2245
2246
2247
2248
2249
2250
2251
2252
2253
2254
2255
2256
2257
2258
2259
2260
2261
2262
2263
2264
2265
2266
2267
2268
2269
2270
2271
2272
2273
2274
2275
2276
2277
2278
2279
2280
2281
2282
2283
2284
2285
2286
2287
2288
2289
2290
2291
2292
2293
2294
2295
2296
2297
2298
2299
2300
2301
2302
2303
2304
2305
2306
2307
2308
2309
2310
2311
2312
2313
2314
2315
2316
2317
2318
2319
2320
2321
2322
2323
2324
2325
2326
2327
2328
2329
2330
2331
2332
2333
2334
2335
2336
233
```


